Universidade de Aveiro Departamento de Engenharia Mecânica
2009

João Pedro Rodrigues dos Santos Dias

Distanciómetro 3D baseado numa unidade laser 2D em movimento contínuo

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica do Dr. Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro

Dedico este trabalho à Inês e à minha família

o júri

presidente	Prof. Dr. Francisco José Malheiro Queirós de Melo Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro
vogal – arguente principal	Prof. Dr. Paulo Miguel de Jesus Dias Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
vogal - orientador	Prof. Dr. Vítor Manuel Ferreira dos Santos Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro

agradecimentos

Ao orientador Vítor Santos pela condução dos trabalhos, ao Eng. Festas por tornar o projecto fisicamente possível e aos colegas de laboratório que sempre se mostraram interessados e prontos a ajudar.

palavras-chave

Microcontrolador, motor passo a passo, “slip ring”, codificador incremental, percepção 3D, nuvem de pontos

resumo

No presente trabalho desenvolveu-se um distanciómetro tridimensional com o auxílio de um laser 2D em rotação contínua. O trabalho inclui a concepção de um protótipo estrutural portátil com um sistema de controlo incorporado, recorrendo à definição de parâmetros de controlo através de um microcontrolador. A reconstrução 3D a partir de um *Laser Range Finder* é obtida através de um grau de liberdade extra conferido por um motor passo-a-passo com controlo de posição. Através de *slip rings* este sistema consegue estar em comunicação contínua com o LRF mesmo estando este em plena rotação, evitando os constrangimentos criados pelas cablagens convencionais. A estrutura concebida é portátil e pode ser montada em posição invertida; a versatilidade da estrutura aumenta o campo de aplicações da mesma.

keywords Microcontroller, stepper motor, slip ring, incremental encoder, 3D perception, points cloud

abstract The present work consists on the conceiving of a tridimensional laser scanning system regarding a 2D Laser Range Finder on continuous rotation. This work includes creating a mobile structural prototype with a built-in control system, using a microcontroller to set the control parameters. 3D reconstruction through a 2D LRF is obtained by having a extra degree of freedom made possible by a stepper motor with position control. The use of slip rings allows to keep constant communication with LRF even when is on continuous rotation, avoiding the usual constrains created by convencional connections. The structure is portable and can be installed on inverted position; the versatility of the structure increases it's application fields.

Índice de conteúdos

Capítulo 1 - Introdução 8
1.1. Objectivos do trabalho 8
1.2. Estado da arte 8
Capítulo 2 - Solução conceptual 14
2.1. Estrutura mecânica 14
2.2. Tracção 16
2.3. Controlo 17
2.4. Alimentação e cablagem 18
Capítulo 3 – Dimensionamento de componentes 22
3.1. Escolha de equipamento 22
3.2. Ajustes mecânicos 26
3.3 Análise estática das peças a conceber 30
Capítulo 4 – Concepção do sistema de controlo 32
4.1. Unidade de controlo 32
4.2. Microcontrolador 34
4.2.1. Configuração inicial do microcontrolador 35
4.2.2. Programação 40
4.2.3. Protocolo de comunicação 50
4.3. Unidade de potência 54
Capítulo 5 - Resultados e conclusões 56
Referências 63
Anexos 65
A1.Main.c 66
A2.Func.c 71
A3.Func.h 74
A4.Var.h 74
A5.Chapa de topo em alumínio – simulação de compressão (100 N) 75
A6.Eixo central em alumínio – simulação de compressão (100 N) 76
A7.Eixo central em alumínio – simulação de torção (1,27 N.m) 77
A8.Fixador em alumínio – simulação de torção (1,27 N.m) 78
A9.Suporte para eixo em alumínio – simulação de compressão (100 N) 79
A10.Suporte inversor em alumínio – simulação de compressão (100 N) 80
A11.Suporte para motor em alumínio – simulação de torção (1,27 N.m) 81
A12.Circuito microcontrolador – esquema eléctrico 82
A13.Circuito potência (L297 + L298) – esquema eléctrico 83
A14.Circuito de alimentação e divisor frequência – esquema eléctrico 84

Capítulo 1

Introdução

Desde há muito que o Homem procura aperfeiçoar a sua percepção do espaço; a visão revela-se insuficiente para a total compreensão do que nos rodeia e a ciência aponta o caminho para vencer os constrangimentos inerentes à condição humana. Será erróneo pensar que o homem vê a três dimensões, no entanto, será correcto dizer que o homem tem percepção tridimensional do espaço, isto indicia que não só da visão se faz a percepção do ambiente envolvente. Do tacto ao som, tudo parece ser uma fonte de informação complementar, do complemento surge informação cada vez mais detalhada. Também a ciência tem desenvolvido sistemas que, em similitude com o Homem, se aperfeiçoam através de processos de complementaridade. Actualmente, algumas soluções para medição de distâncias são realizadas por sistemas laser denominados por LRF (Laser Range Finder). Um feixe de laser será algo semelhante a uma linha unidimensional, o desafio consiste em, partindo dessa mesma linha, ser capaz de obter dados tridimensionais. Uma solução reside em fazer com que esse feixe não seja estático. De facto, se um feixe unidimensional for projectado várias vezes segundo várias direcções, pode-se fazer o reconhecimento superficial do espaço envolvente. Pode-se então dizer que está encontrado um complemento possível para vencer a limitação física do laser no reconhecimento do mundo real. Deste ponto de partida, podem surgir inúmeras soluções ao conferir movimento ao laser.

1.1. Objectivos do trabalho

Este trabalho pretende solucionar uma questão científica e económica, procura-se criar um protótipo que materialize a possibilidade de, através de um LRF scanner bidimensional, conceber uma solução que forneça leituras em tempo real do espaço tridimensional a baixo dos custo. Trabalhos anteriormente realizados já permitiam obter leituras tridimensionais, no entanto, a liberdade de varrimento é reduzida; este trabalho pretende também fornecer uma alternativa no sentido de vencer constrangimentos criados pela cablagem.

1.2. Estado da arte

A percepção de espaços a 3 dimensões assume elevada importância em ambientes industriais e em algumas aplicações comerciais. As unidades LRF apresentam-se como uma solução interessante na reconstrução de ambientes a 3 dimensões, em aplicações de segurança e no reconhecimento de geometrias em linhas de montagem. Existem no mercado dispositivos capazes de fazer reconstrução tridimensional, mas os custos são muito elevados; além disso os *scans* tridimensionais apresentam intervalos de tempo no processo de *scan* pouco apelativos para aplicações de controlo em tempo real [1]. Assim

apresenta-se como uma alternativa viável, apontada por alguns estudos, a aquisição de um *scan* 2D conferindo mais um grau de liberdade através de um sistema mecânico controlado por um servo motor. É sobre esta configuração alternativa em particular e sobre algumas alternativas dentro dessa mesma configuração que se debruçará este trabalho.

Os *scans* 2D têm tempos de varrimento na ordem dos centésimos de segundo, no caso concreto do LRF LMS 200-30106 [Figura 1] o tempo de resposta para uma leitura com amplitude de 180º, com 0.5º de resolução é de 26 milissegundos [5], embora o LRF possa apresentar outra amplitude e outras resoluções.

Figura 1: Laser Sick LMS 200-30106

Antes de partir para a solução mecânica que confere o grau extra de liberdade que garante uma modelação tridimensional do espaço, convém ter um pouco presente o que se tem feito nesta área e que soluções e que limitações apresentam os LRFs bidimensionais.

Parece relevante ter em conta um estudo em particular cujo objectivo seria a detecção de um mini-helicóptero [1], pois este estudo apresenta-se como uma solução de controlo de voo externo interessante que, em simultâneo, analisa algumas dificuldades inerentes ao projecto. Em relação à detecção de objectos através de LRF, parece surgir com naturalidade a seguinte questão: porque não recorrer a uma câmara comum? Uma das mais fortes razões será a versatilidade do LRF em relação a ambientes exteriores e interiores. Embora as câmaras tenham a possibilidade de fazer extracção de objectos por cor e geometria, vêem-se altamente limitadas pelas condições de iluminação. Além disso não possuem um campo de visão tão alargado, o que se revela desvantajoso em aplicações de segurança. E se em ambientes interiores ainda se pode contornar essas dificuldades com o controlo das condições de luminosidade, já no exterior isso torna-se de extrema dificuldade. Pode-se então dizer que embora as câmaras tenham a possibilidade de extracção de objectos por geometria e cor, ambas as possibilidades ficam diminuídas por condições de luminosidade adversas. Por sua vez, o LRF pode fazer extracção de objectos por geometria e dimensão, com um campo de visão mais alargado e dando-nos a distância do objecto sem que este precise de estar iluminado (sistema activo) [3]. A percepção do LRF do espaço é baseada no seu tempo de voo, ou seja, a velocidade de propagação do laser é considerada como uma constante conhecida, assim quando um feixe de laser é emitido e resvala num objecto é reflectido de volta, o tempo entre transmissão e recepção desse sinal é registado sabendo-se assim a distância que separa o obstáculo do LRF. No caso do estudo em questão é usado um LRF comercial da SICK que faz um varrimento

horizontal com um feixes laser. Sabendo a distância de cada ponto juntamente com a informação do deslocamento horizontal em graus, neste caso a deslocação é traduzida pela resolução angular do LRF, pode-se saber a posição de um objecto num plano. Portanto, o LRF bidimensional oferece uma solução muito restrita quando se trata de detectar a presença de um mini-helicóptero, a ascensão em altura do mini-helicóptero seria suficiente para este sair do plano de acção do LRF e assim não ser detectado. Então, surge a solução de acoplar um sistema mecânico que confira mais um grau de liberdade, accionado por um motor passo-a-passo que fará o LRF rodar sobre um segundo eixo. Em última análise tem-se os dados necessários para trabalhar a 3 dimensões: tem-se o raio do laser (r), a direcção de um feixe do LRF (α) e o deslocamento angular do LRF accionado pelo motor (β); a partir destes dados pode-se saber a posição do mini-helicóptero no espaço cartesiano [Figura 2] através das seguintes equações:

$$x = r.cos(\alpha)$$

$$y = r.sin(\alpha).\sin(\beta)$$

$$z = r.sin(\alpha).\cos(\beta)$$

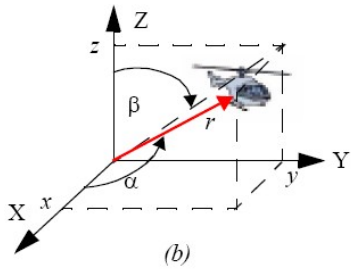


Figura 2: Mini-helicóptero e sua posição no sistema cartesiano

Convém, no entanto, salientar que existem outras configurações possíveis e que dependendo delas ir-se-ão obter equações diferentes; essas configurações variam em relação ao eixo de rotação adicional e à posição do LRF. De seguida, analisam-se essas diferentes configurações e como elas influenciam as nuvens de pontos e a concentração das mesmas. Se se optar por ter o LRF virado para cima e fazendo-o rodar sobre o eixo vertical que trespassa o seu espelho frontal [Figura 3], ter-se-á a formação de uma calote esférica de 360º por 180º, isto após a rotação de 180º do eixo mecânico e se o LRF estiver no modo em que o seu ângulo *scan* é 180º. Esta disposição é ideal para detectar objectos que sobrevoem o LRF ou para recriar tectos de edifícios, dado que o eixo de rotação coincide com a origem dos feixes a nuvem de pontos será mais densa junto hemisfério superior da calote esférica, isto porque a maior concentração de pontos verifica-se junto do eixo de rotação. Ora, quanto mais densa é a nuvem de pontos, melhor se pode definir um objecto na sua geometria, assim pode-se dizer que esta configuração é ideal para detectar objectos que se desloquem por cima do LRF e para recriar a geometria de objectos que se encontrem em frente ao espelho frontal.

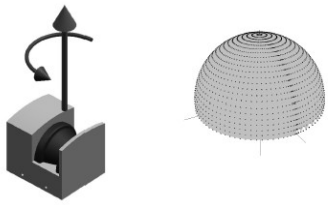


Figura 3: Configuração alternativa "yawing scan top"

Pode-se também, com o mesmo eixo de rotação, optar por outra configuração, se por exemplo se colocar o LRF com o espelho frontal de lado [Figura 4], a nuvem de pontos formada será diferente no que diz respeito à localização das zonas de pontos mais densas, também ao invés de se ter a formação de uma calote poderá-se ter a formação de uma esfera completa após uma rotação de 360º do eixo mecânico. Esta nova disposição apresenta então uma vantagem, na medida em que permite o varrimento completo do tudo o que a rodeia, no entanto isso acontece após uma rotação completa do eixo mecânico. Agora existem duas zonas de densidade de pontos mais elevada, uma zona no hemisfério superior e outra no hemisfério inferior.

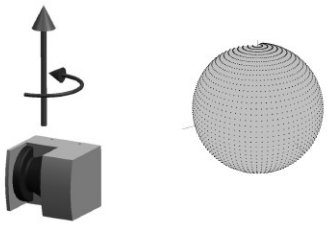


Figura 4: Configuração alternativa "yawing scan"

Existem outras configurações possíveis mas optou-se por enunciar estas duas porque sintetizam as diferenças entre as restantes, ou seja, ou se tem a formação de uma esfera com duas zonas de alta densidade de pontos de medição ou, em alternativa, uma calote esférica com formação de uma zona de alta densidade de pontos de medição. As outras configurações suscitariam os mesmos desenvolvimentos, apenas diferindo a posição relativa da calote e das zonas de alta densidade. O projecto pretende contemplar as duas últimas soluções apresentadas, ou seja, a configuração "yawing *scan* top" [2] e a configuração denominada "yawing *scan*" [2], com primazia para a primeira dado que após meia revolução já tem a calote esférica completa. Foram escolhidas estas duas soluções porque o sistema desenhado para uma será facilmente adaptado para funcionar segundo a outra, dado que ambas as configurações rodam segundo o mesmo eixo (Z), variando apenas a posição relativa do LRF. Este projecto pretende também ser uma alternativa a uma configuração anteriormente testada [Figura 5] na Universidade de Aveiro [6], e pretende-se agora movimento contínuo com rotações ilimitadas. O projecto já existente corresponde a uma configuração semelhante a "yawing scan", no entanto o eixo de rotação não é Z mas X ou Y, se efectuada uma revolução completa pelo motor será constituída uma esfera completa, acontece que o projecto em causa não permite obter uma revolução completa e, logicamente, rotações ilimitadas.

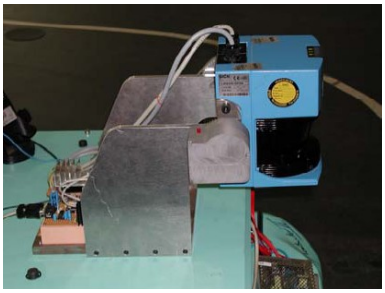


Figura 5: Projecto anteriormente desenvolvido na UA

Um dos desafios quando se usa um LRF2D juntamente com um motor para obter dados 3D prende-se com o cruzamento de dados. Alguns trabalhos apostam no desenvolvimento de algoritmos em sistemas em tempo real que forneçam *timestamps* [4]precisos, podendo assim posteriormente correlacionar os dados. Outra alternativa consiste em anexar dados de posição do LRF com dados do *scan* em tempo real. Este procedimento requer que a taxa de transmissão da posição seja superior à taxa de transmissão do LRF, só assim é possível garantir que dois *scans* consecutivos não estejam afectos à mesma posição.

No caso do modelo LMS200-30106 é enviado um pacote de dados contendo um conjunto de leituras, esse conjunto pode ser, por exemplo, constituído por 361 leituras para um ângulo de *scan* de 180º. Esse pacote será associado a uma posição enviada pelo microcontrolador. Para garantir uma boa correlação entre *scans* e posição do LRF, a taxa de envio de posição do LRF deve ser superior à taxa de transmissão de dados do próprio LRF. Segundo o Teorema de Amostragem de Nyquist [18], a frequência de amostragem (frequência de envio de posição do LRF) deve ser maior que o dobro da frequência do sinal a ser amostrado (frequência de transmissão de dados do LRF). O *scan* decorrerá a uma frequência de aproximadamente 38 Hertz (26 milissegundos por scan), logo utilizar-se-á uma taxa de envio de posição do LRF de aproximadamente 76 Hertz (13 milissegundos).

Alguns trabalhos [15] propõem expressões algébricas para correcção de erros, que pode ter várias origens:

- erros de instalação gerados pelo facto de a origem do feixe de laser não coincidir com o centro de rotação do LRF;

- erros de alcance relacionados com condições materiais, como a superficie dos objectos de incidência ou a humidade do ar;

- erros de ângulo de *scan* criados a quando da aceleração da unidade;

$$x_{ij} = r_{ij} \cos\beta_j \cos\alpha_i$$
$$y_{ij} = r_{ij} \sin\beta_j \cos\alpha_i$$
$$x_{ij} = r_{ij} \sin\alpha_i$$

$$x_{ij} = (\rho_{ij} + \Delta\rho)\cos\alpha_i \cos(\beta_j + \varphi) + l\cos(\beta_j + \varphi)$$
$$y_{ij} = (\rho_{ij} + \Delta\rho)\cos\alpha_i \sin(\beta_j + \varphi) + l\sin(\beta_j + \varphi)$$
$$z_{ij} = (\rho_{ij} + \Delta\rho)\sin\alpha_i$$

Como se pode verificar nas expressões acima, partindo de coordenadas esféricas são introduzidas expressões de correcção. Sendo que r é a distância ao alvo, ß é o ângulo de *scan* do LRF e α é o ângulo do feixe laser. Como se pode averiguar, o erro de alcance Δρ afecta directamente a medida real da distância ao alvo, o erro de instalação l do LRF vem também afectado do ângulo de *scan* ß e do seu erro φ.

Em termos comerciais já existem dispositivos capazes de realizar aquilo a que este trabalho se propõe. Por exemplo, atente-se ao LRF de alta definição Velodyne HDL-64E S2 [Figura 6].

Specifications	
Sensor:	• 64 lasers/detectors • 360 degree field of view (azimuth) • 0.09 degree angular resolution (azimuth) • 26.8 degree vertical field of view (elevation) - +2° up to -24.8° down with 64 equally spaced angular subdivisions (approximately 0.4°) • <2 cm distance accuracy • 5-15 Hz field of view update (user selectable) • 50 meter range for pavement (~0.10 reflectivity) • 120 meter range for cars and foliage (~0.80 reflectivity) • >1.8 M points per second • <0.05 milliseconds latency
Laser:	• Class 1 - eye safe • 4 x 16 laser block assemblies • 905 nm wavelenth • 5 nanosecond pulse • Adaptive power system for minimizing saturations and blinding
Mechanical:	• 12V input (16V max) @ 4 amps • <29 lbs. • 10" tall cylinder of 8" OD diameter • 300 RPM - 900 RPM spin rate (user selectable)
Output:	• 100 MBPS UDP Ethernet packets

Figura 6: Especificações do modelo HDL-64E S2

Esta unidade compacta de 13 quilogramas tem a particularidade de ter 64 emissores laser, contrastando com lasers como o SICK LMS200-30106 que têm apenas um feixe emissor. Capaz de fornecer 1,8 milhões de pontos por segundo com uma amplitude horizontal de 360º e vertical de 26,8º é uma unidade com características ideais para condução autónoma, tendo sido já utilizado na competição "*DARPA Challenge"*. Talvez a principal limitação seja a amplitude vertical de 26,8º que leva a que este dispositivo não seja capaz de criar um nuvem de pontos em forma de calote esférica, ou muito menos seja capaz de reconstruir uma esfera completa. Tem como grande vantagem os níveis de resolução, tendo uma resolução angular de 0,09º horizontal e 0,4º vertical.

Capítulo 2

Solução conceptual

A solução conceptual materializa uma solução genérica para o problema e traça as linhas orientadoras do projecto até ao seu estado final. De seguida são apresentadas as ideias que estiveram na origem do projecto, não sendo soluções rigorosas são as primeiras abordagens ao problema.

2.1. Estrutura mecânica

O primeiro passo consiste em criar uma estrutura [Figura 7] que permita na sua parte superior fixar o LRF e na sua parte inferior conter vários componentes necessários ao sistema. A ligar as duas partes deve existir um eixo que poderá rodar livremente.

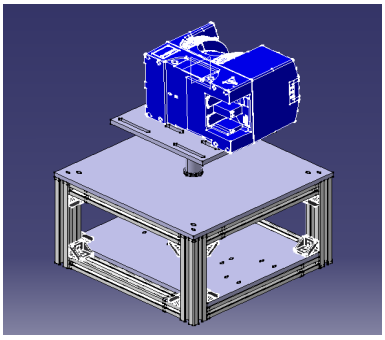


Figura 7: Estrutura conceptual

Na base será criada uma caixa delimitada por perfis de alumínio e limitada, no seu topo, por uma chapa de alumínio que terá uma perfuração que permite ser atravessada pelo eixo. O eixo será solidariamente ligado a uma plataforma à qual será acoplado o LRF. A plataforma [Figura 8] será concebida prevendo a possibilidade de fixar o LRF em posição lateral ou virado para cima. Assim será possível o funcionamento segundo as duas configurações previamente descritas, "yawing *scan* top" e "yawing *scan*". As furações serão longas para permitir o ajuste de posição do LRF. O ajuste de posição visa salvaguardar a possibilidade de centrar a origem do feixe laser do LRF com o centro do eixo.

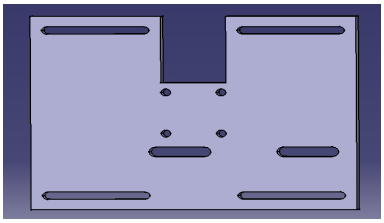


Figura 8: Plataforma de fixação do laser

Para permitir ao eixo rodar livremente com níveis de atrito reduzidos recorre-se a rolamentos axiais de esferas. Através de sucessivos estrangulamentos no diâmetro do eixo [Figura 9] será possível que o mesmo se apoie sobre os rolamentos.

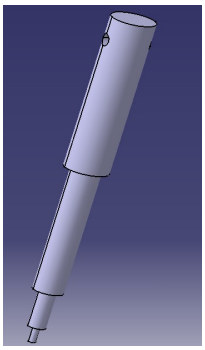


Figura 9: Eixo de diâmetro variável

A estrutura poderá ser suspensa do avesso e para evitar que o eixo se separe da caixa deverá ser criado um anel com furo passante [Figura 10] que atravessará o eixo.

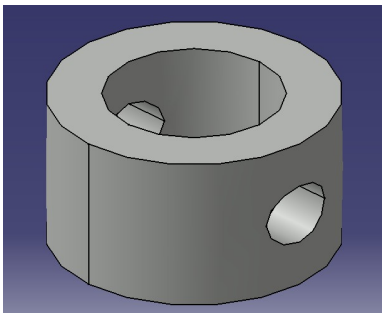


Figura 10: Anel de furo passante

Para que, em posição invertida, o eixo possa rodar sobre um rolamento e esse mesmo rolamento se encontre num encaixe justo, será criada uma chapa [Figura 11] que ficará situada por baixo da chapa superior. Esta chapa é desenhada com furação à medida do eixo e do rolamento.

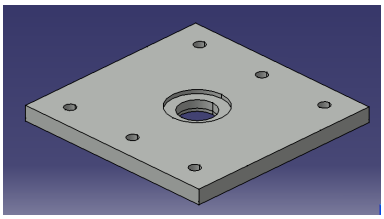


Figura 11: Chapa para encaixe de eixo e rolamento

Da criação destas duas peças, chapa e anel, resulta um sistema de encaixe [Figura 12] que permite que, em posição invertida, o anel comprima o rolamento axial mantendo o eixo fixo através de um parafuso que atravessará o anel e o eixo em simultâneo.

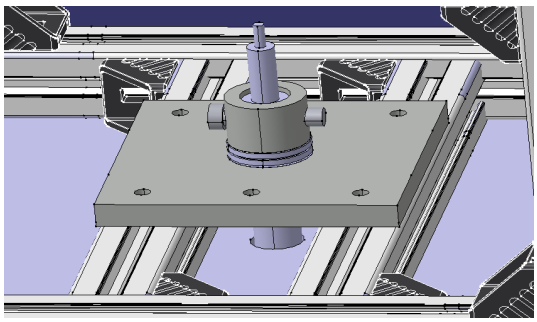


Figura 12: Pormenor do sistema de inversão

2.2. Tracção

O desenvolvimento de um sistema de tracção que transmita o movimento para o eixo central deve ser pensado de forma a minimizar desgastes e a economizar espaço. Dado que o espaço por baixo do eixo deve ser reservado para outras aplicações, não será possível realizar uma transmissão directa entre motor e eixo através de um acoplamento. A solução passará por um sistema de transmissão indirecto.

Na escolha do motor há alguns cuidados a ter em conta dado que a tipologia e as suas capacidades serão decisivas na qualidade do projecto final. No que diz respeito à tipologia, sendo o objectivo deste projecto o funcionamento em movimento contínuo, um motor DC poderia ser uma boa solução. Embora para funcionamento contínuo um motor DC seja uma solução viável, o seu uso para posicionamento preciso requer um servo-sistema adicional.

Os servo motores [12] são amplamente utilizados em aplicações de robótica, no entanto, os custos associados são relativamente elevados. Para este projecto opta-se pela implementação de um motor passo a passo [13]. O motor passo a passo permite obter um funcionamento próximo ao do motor DC, embora funcione com um sistema de passos angulares sucessivos; para elevadas frequências o seu movimento é em tudo análogo ao movimento contínuo. Por sua vez, os servo motores tem custos associados mais elevados porque trazem incorporados controladores e codificadores.

Escolhido o tipo de motor, é necessário analisar as condições para o integrar na estrutura. Relembrando que o eixo central está em posição vertical, a abordagem mais simples para permitir a transmissão será colocar o motor em posição vertical também. A furação para fixação dos motores encontra-se na face que contém o veio; não havendo fixação na base, será necessário desenhar uma peça que permita a fixação do motor em posição vertical. A peça projectada [Figura 13] envolve o motor e tem um furo central bastante largo para permitir, se necessário, desanexá-la do motor sem remover a polia que estará ancorada no seu veio.

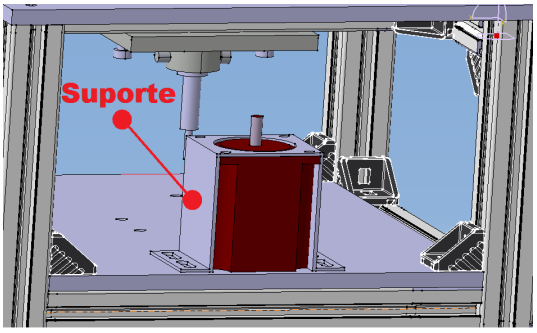


Figura 13: Suporte para fixação de motor em posição vertical

Não será possível uma transmissão directa entre veio do motor e eixo. A questão resolve-se recorrendo a um sistema de transmissão constituído por duas polias e uma correia [Figura 14]. Esta opção salvaguarda a possibilidade de alterar a relação de transmissão. A transmissão através de uma correia cria menos desgaste do que a transmissão directa entre rodas dentadas, desgaste esse que poderia levar à perda de precisão ou até à necessidade de substituição. Usando uma correia pode-se fazer um dimensionamento de acordo com o espaço que se pretenda ter entre eixo e motor sem ter de alterar, ou apenas alterando ligeiramente, a dimensão das polias.

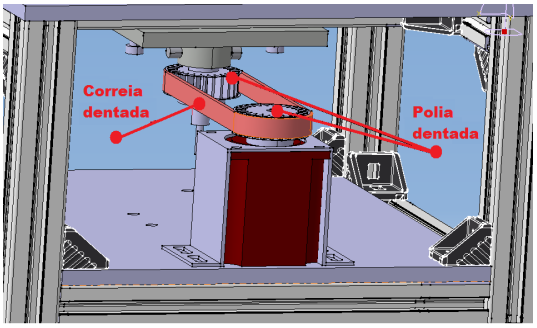


Figura 14: Montagem do sistema de transmissão

2.3. Controlo

A criação de um sistema de controlo específico para este projecto prende-se com um objectivo primordial: a possibilidade de controlar o motor no que diz respeito à sua velocidade e posição. A determinação da posição real do LRF é feita através de um codificador.

Existem codificadores absolutos e incrementais, mas opta-se pelos codificadores incrementais dado que são mais económicos e visto que com estes continua a ser possível reconhecer com precisão o posicionamento angular. Os codificadores incrementais têm, pelo menos, três canais, os canais A, B e Z. Os canais A e B correspondem a ranhuras desfasadas entre si ao longo do perímetro do disco. O desfasamento existente entre canal A e B permite saber o sentido de rotação do veio. O canal Z consiste numa única ranhura. Cada vez que é emitido um sinal do canal Z significa que uma volta foi completada.

Este equipamento deve ser colocado na estrutura de forma a que seja possível ligar eixo central ao veio do mesmo. Como é necessário garantir uma ligação estável entre eixo do codificador e eixo central, criou-se uma estrutura [Figura 15] para ligar a chapa superior e inferior. Ligando solidariamente a chapa superior e inferior é garantido uma minimização das translações de uma em relação à outra. Isto, por sua vez, garante que não haverá desalinhamentos significativos que possam danificar o codificador. A furação para fixação encontra-se na superfície onde se situa o veio.

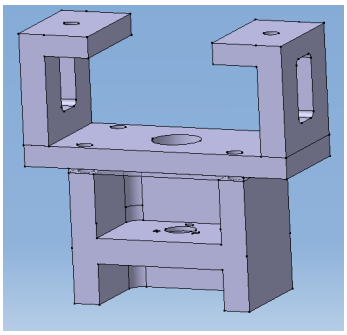


Figura 15: Suporte para codificador

O veio do codificador será ligado através de um acoplamento rígido ao extremo do eixo central. A opção pelo acoplamento rígido prende-se com a necessidade de precisão; um acoplamento flexível admitiria algum desfasamento, ainda que ligeiro, entre a posição do eixo e a posição do veio do codificador. É também conveniente salientar que o uso de um acoplamento rígido exige mais cuidado no projecto. A existir algum desalinhamento entre eixo e veio, o acoplamento rígido poderá danificar bastante mais o codificador do que se se tratasse de um acoplamento flexível. Obtem-se uma montagem [Figura 16] eficaz entre veio e eixo através destes dois componentes, suporte e acoplamento rígido.

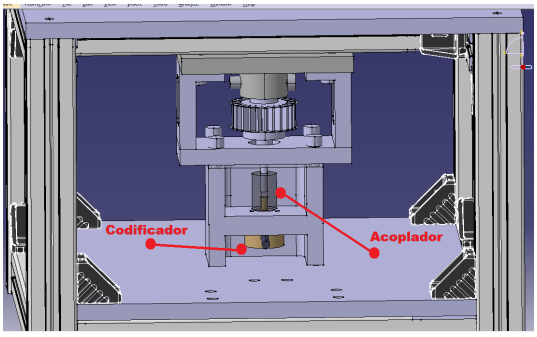


Figura 16: Montagem de codificador

2.4. Alimentação e cablagem

Neste projecto em concreto é necessário ter dois níveis de tensão eléctrica, 24 e 5 Volt. Os 24 Volt são necessários para alimentação do LRF e do motor, ao passo que os 5 Volt servem para alimentar o circuito microcontrolador. Os 5 Volt para o circuito do microcontrolador podem ser obtidos através de reguladores de tensão incorporados no circuito. É necessário ter em consideração duas possibilidades de funcionamento do

sistema, em espaços exteriores e interiores. Para espaços interiores assume-se a existência de tomadas, logo será preciso prever um transformador que transforme os 220-240 VAC para 24 VDC. A actuação em espaços exteriores requer a autonomia do sistema. A incorporação de 2 baterias de 12 Volt ligadas em série, permite obter os 24 Volt para alimentar o equipamento sem recurso a tomadas.

As baterias [Figura 17] são instalados nos espaços previamente destinados para o efeito.

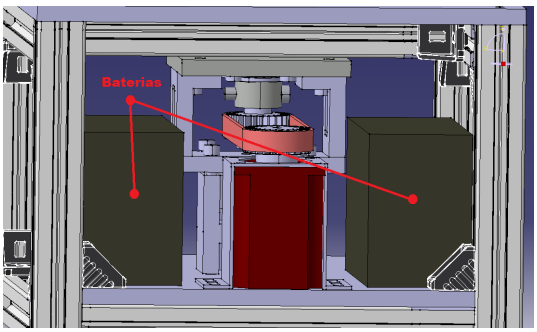


Figura 17: Montagem das baterias

O transformador [Figura 18], tal como toda a estrutura, deverá funcionar em posição invertida, pelo que mais uma peça será desenvolvida para o anexar à base da caixa. Os movimentos de translação serão constrangidos por peças em acrílico colocadas em faces opostas do transformador.

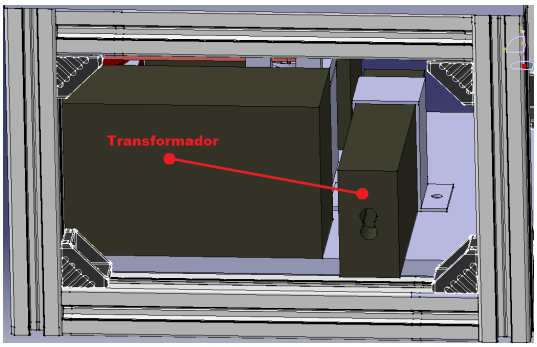


Figura 18: Montagem do transformador

A diversidade de equipamentos pressupõe um número de ligações físicas ainda maior. A limitação de espaço ou a necessidade de liberdade de rotação levam a que as ligações, embora imprescindíveis, se revelem um problema que tem de ser pensado "à priori".

No âmbito deste projecto existe uma preocupação central no que diz respeito a cablagem. Como permitir que o LRF rode indefinidamente sobre um eixo sem que os seus cabos de alimentação e comunicação se enrodilhem e danifiquem a si mesmos e ao equipamento circundante?

Uma das soluções inicialmente apontadas foi a incorporação de um sistema WLAN (Wireless Local Area Network). Um sistema de comunicação sem fios por frequências rádio poderia resolver o problema de comunicação mas não o problema de alimentação. O problema continua a ser uma realidade porque serão precisos cabos para fornecer alimentação para o LRF. A solução passa por perceber como se pode ultrapassar os limites de ligações físicas através de meios, também eles, físicos.

Existe a tecnologia desenvolvida ao nível dos motores. Os motores de escovas possuem escovas que são responsáveis por transmitir energia eléctrica ao rotor. No entanto, estas aplicações desenhadas para motores são grosseiras se se pensar em comunicar dados, a comunicação de dados não admitiria os níveis de ruído gerados pelos contactos intermitentes ou pela fricção.

A literatura [1][3][16] fornece indicações no sentido de que a comunicação de dados e transmissão de potência será viável através de dispositivos denominados por "Slip rings". Após pesquisa por informação sobre este produto, percebe-se que a tecnologia consiste numa parte rotativa e uma parte estática, existindo duas configurações possíveis, sendo que a parte estática e rotativa comutam. Uma das configurações consiste em anéis que rodam solidariamente com o dispositivo ao qual estão fixos, e uma parte estática onde se encontram contactos do tipo escova [Figura 19]; em outras unidades são as escovas que rodam e os anéis encontram-se estáticos [14].

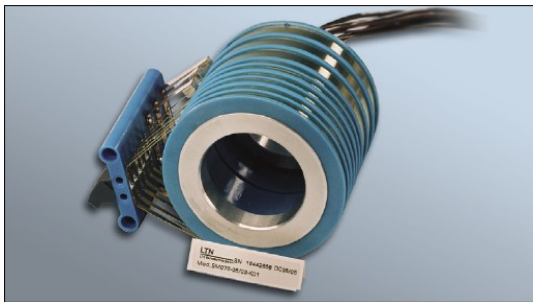


Figura 19: Slip ring com furo interior sem encapsulamento

Esta tecnologia é fundamental para o problema, dado que existem unidades compatíveis com diversos protocolos de comunicação, como Ethernet ou Firewire, que, ao mesmo tempo, conseguem transmitir sinais de potência. A solução para transmissão de potência para alimentação do LRF passaria, muito provavelmente, por este tipo de dispositivos. A capacidade de transmitir dados prende-se com os baixos níveis de ruído destas unidades. Baixos níveis de ruído são resultado de contactos de alta qualidade concebidos em materiais preciosos tais como o ouro. No âmbito deste projecto, procura-se em particular unidades com furação interior e encapsuladas [Figura 20]. A furação deve possibilitar que a unidade seja atravessada e acoplada ao eixo em rotação; o encapsulamento visa garantir que não ocorre entrada de detritos nos contactos.

Figura 20: Slip rings com furo interior e encapsulamento

A montagem [Figura 21] deste dispositivo fica simplificada pela sua concepção, sendo apenas necessário atravessá-lo pelo eixo de rotação e fixá-lo ao mesmo. A fixação é feita através de parafusos dispostos radialmente no furo interior que comprimem o eixo.

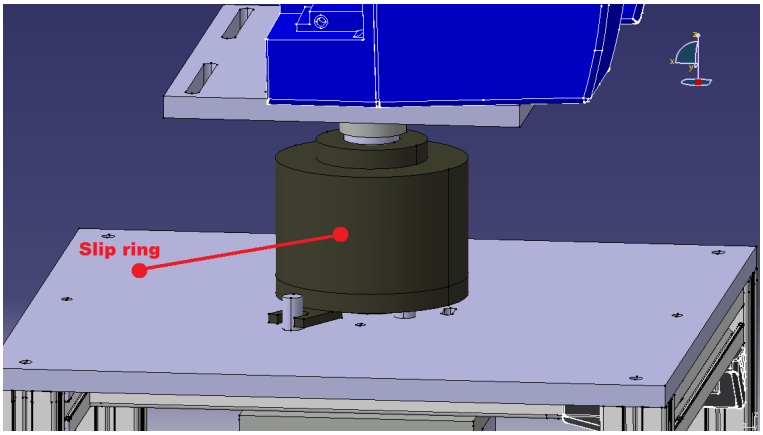


Figura 21: Fixação do slip ring no eixo de rotação

Capítulo 3

Dimensionamento de componentes

Após a apresentação das ideias gerais sobre a estrutura segue-se a fase de dimensionamento dos diversos componentes.

3.1. Escolha de equipamento

Para escolha do motor, o primeiro passo é calcular o binário que a estrutura precisará e, de seguida, escolher um motor com um binário superior. Considere-se, por motivos de simplificação, que a estrutura pode ser equiparada à soma de 3 elementos distintos [Figura 22]. Assuma-se também que, em todos eles, o eixo de rotação atravessa o centro de massa que será coincidente com o centro de gravidade. Os elementos são o eixo cilíndrico (cilindro), a base de suporte para o LRF (paralelipípedo) e o próprio LRF (cubo).

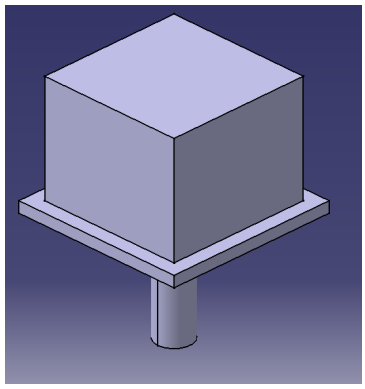


Figura 22: Simplificação do sistema inercial

Calcule-se as inércias [7]para os 3 elementos distintos:

$$I_g = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

Eixo cilíndrico

Apenas se pretende que o eixo cilíndrico [Figura 23] rode segundo o eixo vertical Z que coincide com o seu eixo central, pelo que apenas se torna necessário calcular a inércia segundo Z. O suporte será construído em alumínio de densidade 2700 Kg/m³. O eixo deverá ter aproximadamente 200 milímetros de comprimento e 25 milímetros de diâmetro.

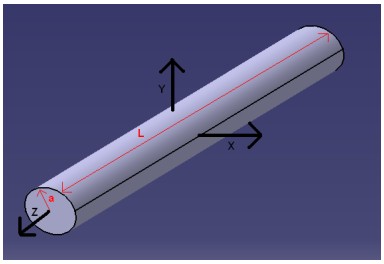


Figura 23: Referencial para eixo cilíndrico

$$I_{zz} = \frac{M.R^2}{2} = \frac{\rho.V.R^2}{2} = \frac{\rho.\pi.R^2.L.R^2}{2}$$

$$\frac{\rho.\pi.L.R^4}{2} = \frac{2700 \times \pi \times 0{,}200 \times 0{,}0125^4}{2} = 0{,}0000207\ Kg.m^2$$

Base de suporte para o LRF

A base de suporte [Figura 24] terá aproximadamente a dimensão do LRF (155x185) com 10 milímetros de espessura.

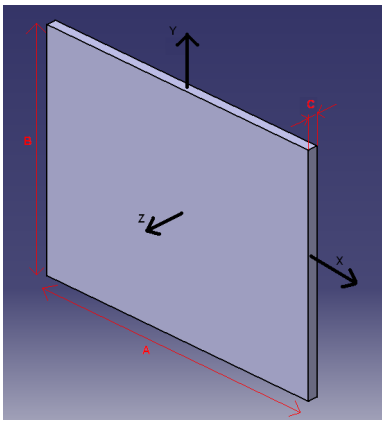


Figura 24: Referencial inercial para base de suporte

$$I_{zz} = \frac{M.(A^2+B^2)}{12} = \frac{\rho.V.(A^2+B^2)}{12} = \frac{\rho.A.B.C.(A^2+B^2)}{12}$$

$$I_{zz} = \frac{2700 \times 0{,}155 \times 0{,}185 \times 0{,}01 \times (0{,}155^2+0{,}185^2)}{12} = 0{,}00375\ Kg.m^2$$

LRF

A massa do LRF é, segundo informação do fabricante, de 4,5 Kg.

$$I_{zz} = \frac{M.(A^2 + B^2)}{12} = \frac{4.5 \times (0.155^2 + 0.185^2)}{12} = 0.02184\ Kg.m^2$$

De seguida calculam-se os momentos dinâmicos de cada elemento. Somando todos obtém-se o binário mínimo necessário para conseguir mover a estrutura a partir do repouso.

$$H_g = I_g \times \omega$$
$$M_b = \dot{H}_g = I_g \times \alpha + \omega \times H_g$$

$$Hg = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \omega_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \omega_z . I_{zz} \end{bmatrix}$$

$$M_b = K_g = I_g \times \alpha + \omega \times H_g = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \alpha_z \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \omega_z \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \omega_z . I_{zz} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \alpha_z . I_{zz} \end{bmatrix}$$

Pode-se concluir que apenas se tem de projectar a aceleração angular que se pretende, pois será esse o factor que ditará o binário necessário para vencer a inércia do sistema quando este está parado. Pretende-se que o sistema forneça uma rotação de 5 revoluções por segundo, o que equivale a 300 r.p.m.

Considere-se que se atingirá essa velocidade ao fim de 5 segundos.

$$\alpha = \frac{\omega}{\Delta t} = \frac{300 \times 2.\pi}{60 \times 5} = 6{,}28\ rad/s^2$$

$$M_{b.eixo} = \alpha_z . I_{zz} = 6{,}28 \times 0{,}00002 = 0{,}00013\ N.m$$
$$M_{b.base} = \alpha_z . I_{zz} = 6{,}28 \times 0{,}00375 = 0{,}02355\ N.m$$
$$M_{b.laser} = \alpha_z . I_{zz} = 6{,}28 \times 0{,}02184 = 0{,}13716\ N.m$$

$$M_{b.total} = M_{b.eixo} + M_{b.base} + M_{b.laser} = 0{,}00013 + 0{,}02355 + 0{,}13716 = 0{,}161\ N.m$$

Das soluções pesquisadas no mercado a mais interessante é a de um motor com um binário de 1,27 N.m, embora este esteja bastante sobre-dimensionado é uma das soluções mais económicas encontrada. Trata-se do motor SANYO DENKI 103H7126-0440 [Figura 25] que admite uma frequência de pulsos máxima de 1000 P.P.S. (pulsos por segundo) para passos de 1.8º ou 2000 P.P.S. para passos de 0.9º.

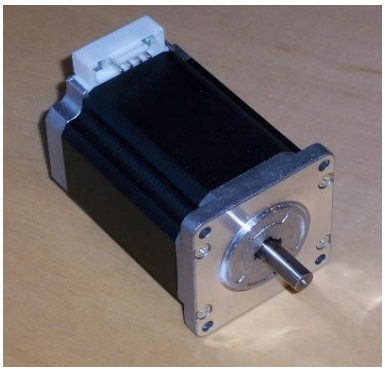


Figura 25: Motor passo a passo Sanyo Denki 103H7126-0440

A condição prioritária na escolha de um codificador incremental é a sua relação de pulsos por revolução (P.P.R). Opta-se pelo codificador Kluber 05.2400.1122.1024[Figura 26] com 1024 P.P.R.

Figura 26: Codificador incremental Kluber

A uma relação de 1024 P.P.R. corresponde uma resolução angular de:

$$\frac{360^{\circ}}{1024} \approx 0.35^{\circ}$$

Sabe-se que o LRF SICK LMS 30106 pode obter leituras para uma distância máxima de 20 metros. Sabendo que a resolução angular é de 0.35º:

$$2\pi \rightarrow 360^{\circ}$$
$$x \rightarrow 0{,}35^{\circ}$$

$$x \approx 6\text{x}10^{-3}\, rad$$
$$20 \times 6\text{x}10^{-3} = 0{,}12\, m$$

Pode-se dizer que para objectos detectados pelo LRF à máxima distância, o codificador fornece uma resolução horizontal de 0,12 metros entre dois pontos consecutivos.

Pretende-se a aquisição de um transformador que converta 220-240 VAC das tomadas em 24 VDC para o LRF e motor. É necessário calcular a corrente necessária para se poder proceder à escolha do transformador.

LRF:

$$P = V \,.\, I \Leftrightarrow I = \frac{P}{V} = \frac{35\,Watts}{24\,Volts} = 1{,}5\,A$$

O motor precisará de 2 Ampere e o circuito de 0,5 Ampere, ao todo são necessários 4A. Opta-se pelo modelo AS-120P-24 [Figura 27] com 5A e 120W de potência.

Figura 27: Trasformador AS-120P-24

As baterias para este projecto são o modelo Stecco Saphir 65 de 12 Volt disponíveis no DEMUA.

O "slip ring" serve para alimentar o LRF e proceder à comunicação de dados. A corrente associada aos protocolos de comunicação é desprezável, logo a corrente necessária prende-se maioritariamente com os requisitos da alimentação do LRF. Como já foi calculado, o LRF necessita de 1,5 Ampere. São necessárias 8 conexões, 5 para dados e 3 para alimentação. O dispositivo deve ter a capacidade de atingir as 300 rpm. Dado que os modelos economicamente mais acessíveis são modelos estandardizados, opta-se pelo modelo Kyh025-12 [Figura 28] com 12 ligações de 5 Ampere, capacidade para atingir as 500 rpm e diâmetro interior de 25.4 milímetros.

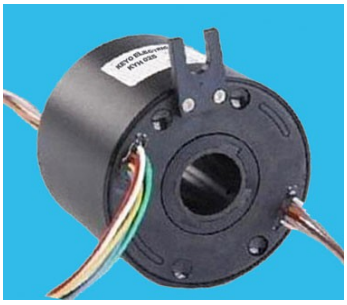


Figura 28: Slip ring modelo Kyh025

3.2. Ajustes mecânicos

Dado o diâmetro do eixo pode-se dimensionar os rolamentos axiais de esferas. Devem-se procurar rolamentos cujo diâmetro interno seja inferior ao diâmetro do eixo. Opta-se pelo modelo 51103 [Figura 29]com 17 milímetros de diâmetro interno e diâmetro

mínimo de apoio de 25 milímetros. A capacidade de carga dinâmica é de 11,4 KiloNewton, a carga a que o o rolamento estará sujeita não superará os 100 Newton.

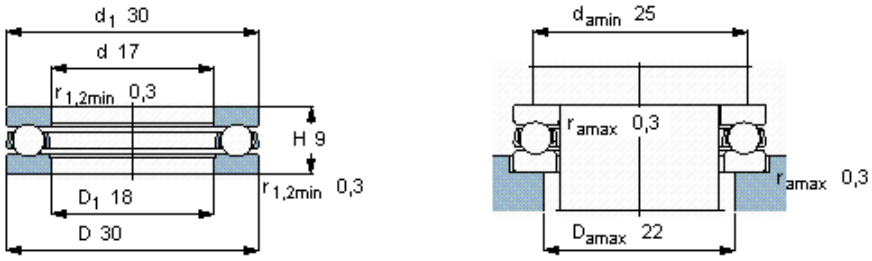


Figura 29: Dimensões do rolamento axial de esferas 51103

Após escolha do rolamento axial é possível também dizer que o primeiro estrangulamento [Figura 30] do eixo será de 25 para 17 milímetros. Para garantir que o eixo não fica com folgas indesejáveis ao atravessar o rolamento, o ajuste veio-furo deve ser executado segundo tolerâncias pré-estabelecidas[10].

.

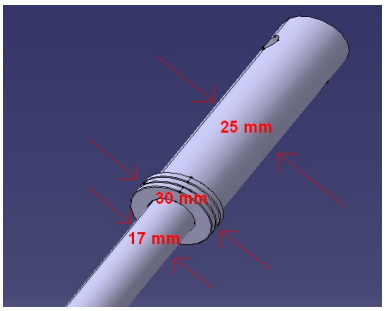


Figura 30: Estrangulamento de eixo com encaixe de rolamento

Será possível controlar um motor passo a passo à taxa máxima de 2000 P.P.S. com um passo angular de 0.9º.

$$Velocidade = \frac{60 \times 2000 \times 0.9}{360} = 300\,rpm$$

Dado que a velocidade pretendida pode ser obtida directamente do veio do motor, a relação de transmissão será unitária. Seria possível ter velocidades inferiores com relações de transmissão diferentes e, com isso, conseguir um maior binário no eixo de rotação que suporta o LRF.

Determinada a relação de transmissão, o dimensionamento das polias deve ser feito tendo em conta que o veio do motor tem um diâmetro de 6,4 milímetros. Assim, opta-se por uma polia com furação de 6 milímetros que poderá ser alargada para receber o veio do motor. Entre as polias com a furação pretendida dever-se-à escolher uma que tenha um número elevado de dentes porque isso traduz-se numa melhor qualidade de transmissão e menor desgaste. Opta-se por polias de 24 dentes 286-5679 com 5 milímetros de passo.

O passo seguinte consiste em realizar o dimensionamento da correia para as polias escolhidas. Para tal é necessário conhecer o diâmetro das polias e o espaçamento entre os centros das polias. A correia deve ser suficiente para compensar os diâmetros da polias sem, no entanto, ser demasiado grande para sair fora da base projectada. Sabendo que eixo se encontrará centrado numa base quadrada de 304 milímetros [Figura 31] e que o veio do motor está inserido no centro de um face quadrada de 56 milímetros, pode-se determinar um espaçamento entre eixo e veio ao qual corresponderá o espaçamento entre os centros das polias.

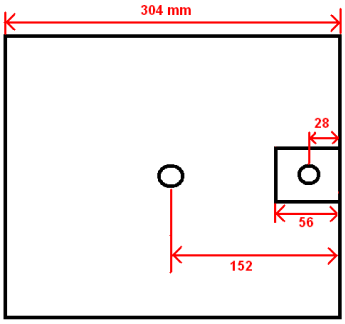


Figura 31: Posicionamento do motor na estrutura (visto de cima)

A dimensão da correia deverá corresponder à soma de metade do perímetro de cada uma das polias mais duas vezes a distância entre eixos [Figura 32].

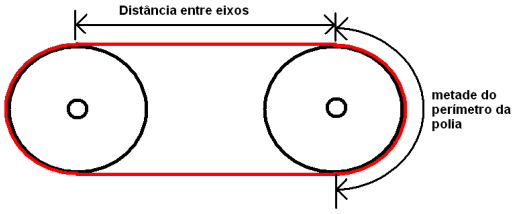


Figura 32: Dimensionamento da correia

$$Comprimento_{correia} = \frac{Perímetro_{polia1}}{2} + \frac{Perímetro_{polia2}}{2} + 2 \times Distância_{entre.eixos} \Leftrightarrow$$

$$\Leftrightarrow Comprimento_{correia} = \frac{\pi \times 37.35}{2} + \frac{\pi \times 37.35}{2} + 2 \times (152 - 28) = 365.3\, mm$$

Estes cálculos têm alguns pressupostos e que convém referir:

→ Considera-se que a zona de contacto da correia com a polia ocorre em metade do perímetro da polia, o que não é totalmente exacto.

→ Como a relação de transmissão é unitária não se torna necessário ter em conta o ângulo que se formaria na correia caso as polias fossem de diâmetros diferentes

→ A distância entre eixos tem de ser superior à distância mínima admissível entre polias, essa distância mínima admissível é soma dos raios exteriores, e não os raios da zona dentada, das duas polias

Sabendo agora o diâmetro do furo central das polias, é possível criar mais um estrangulamento no eixo. O último estrangulamento que servia para atravessar os rolamentos axiais era de 17 milímetros; para que seja possível inserir e remover o eixo na estrutura o diâmetro tem de decrescer sucessivamente. O estrangulamento será de 17 milímetros para 10 milímetros; embora o diâmetro do furo central da polia seja de 6 milímetros, é possível alargá-lo para 10 milímetros.

Dado que o codificador escolhido tem um veio de 4 milímetros, o acoplador a adquirir deve ter um diâmetro interno de 4 milímetros. O modelo Ruland MCLX-4-4-A [Figura 33] é um acoplamento rígido com as dimensões desejadas.

Figura 33: Acoplador rígido Ruland MCLX-4-4-A

Para realizar a transmissão do movimento do eixo para o codificador é necessário proceder a um último estrangulamento. Dado que o acoplador tem um furo interior de 4 milímetros o eixo deve assumir esse mesmo valor. O eixo terá então 3 estrangulamentos [Figura 34].

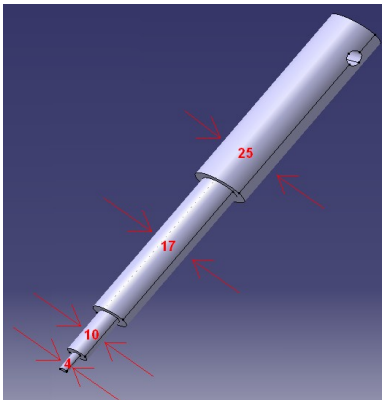


Figura 34: Eixo central

3.3. Análise estática das peças a conceber

De seguida apresentam-se algumas simulações feitas recorrendo ao auxílio do software CATIA, em particular ao CATAnalysis que possibilita simular os esforços e obter uma previsão da reacção dos materiais a esses mesmos esforços.

Antes de mais, convém salientar que as peças foram testadas a dois esforços, um esforço de compressão, causado maioritariamente pelo peso do LRF, e um momento torçor, causado pela rotação do LRF. Como esforços temos a compressão vertical gerada pelo peso do laser, mas não só, outros elementos, como o eixo rotativo e a mesa do LRF, podem também ser acrescidos na criação de esforços de compressão. O LRF pesa 4,5 Kg; assim, opta-se, por segurança, simular os esforços de compressão a 10 Kg. Esta abordagem apresenta-se como conservadora dado que este peso é distribuído por outras peças, mas permite obter conclusões seguras sobre os deslocamentos criados pelos esforços. O momento torçor gerado está relacionado com a inércia do sistema constituído por laser, mesa do LRFe eixo central. O momento necessário para vencer essa inércia já foi calculado, rondando os 0.16 N.m, no entanto opta-se por simular para o binário máximo do motor que é de 1.27 N.m, traduzindo-se mais uma vez numa análise conservadora mas segura. As imagens referentes às simulações encontram-se em Anexo.

Chapa de topo

É possível consultar em anexo [A5] a simulação para a chapa de topo, esta chapa suporta o peso do laser, do eixo central, da mesa do LRF, do fixador da mesa do LRF ao eixo central e do rolamento axial. Como é referido, as compressões verticais foram simuladas para um peso de 10 Kg; dado que os deslocamentos obtidos para esta chapa em alumínio eram pequenos e satisfatórios conclui-se que, para a peça em questão, a opção de simular a compressão a 10 Kg e de conceber a peça em alumínio eram opções válidas.

Eixo central

A peça que se segue será a peça vital no funcionamento da estrutura, o eixo central é responsável por transmitir a rotação ao LRF e sustentá-lo, estando sujeito tanto a compressão como a torção. São necessárias 2 simulações, uma de compressão [A6] e outra de torção [A7], para o eixo constituído por alumínio. Dada a importância deste peça, ainda se ponderou a concepção da mesma em aço, no entanto, através simulações, conclui-se que tal não será necessário. Os deslocamentos não são significativos e o alumínio apresenta-se como uma solução mais leve.

Fixador

O fixador é a peça que permite que a mesa LRF rode solidariamente com o eixo central; esta peça está sujeita a esforços de torção e será concebida em alumínio. Os resultados da simulação [A8] demonstram deslocamentos máximos desprezáveis na ordem das décimas de micrómetro.

Suporte para eixo

Esta peça foi concebida para ser atravessada pelo eixo central, possui também um sulco que permite o acoplamento eficaz do rolamento axial que auxilia a rotação do eixo. Esta peça em alumínio estará sujeita a compressão [A9] quando toda a estrutura estiver em posição invertida. Além de auxiliar no alinhamento do eixo, serve também de suporte do LRF quando em posição invertida, é nessa mesma posição que o esforço de compressão será maior.

Suporte inversor

O suporte inversor tem como principal função ser acoplado ao eixo de forma a permitir que, ao inverter a estrutura, o eixo não deslize por entre os seus apoios. Os esforços serão de compressão [A10] e estarão localizados nos furos da peça que estão destinados a ser atravessados por um pequeno veio que por sua vez atravessará o eixo.

Suporte para motor

Esta é a peça concebida para permitir manter o motor em posição vertical, sendo que a sua maior ou menor estabilidade pode afectar a qualidade de transmissão. Para garantir que não existem prejuízos significativos no sistema de transmissão, simula-se [A11] o comportamento da peça quando sujeita a esforços de torção.

Capítulo 4

Concepção do sistema de controlo

Para operacionalizar o dispositivo torna-se necessário conceber um sistema com realimentação que possibilite a aquisição, processamento e envio de dados. Os microcontroladores são amplamente usados em produtos comerciais, representando uma solução estável de controlo com entradas e saídas. De seguida enumeram-se algumas características e funcionalidades dos microcontroladores [8]:

- Programável
- Entradas e saídas digitais e analógicas
- Memória interna para armazenamento de dados e programa
- Porta de comunicação série
- Temporizadores
- Interrupts
- Gerador de pulsos (*Pulse Width Modulation)*

O microcontrolador não é o único elemento do sistema mas é, com certeza, o elemento chave no todo que é a unidade de controlo.

4.1. Unidade de controlo

A unidade de controlo deve ser entendida como a súmula de todos os elementos físicos necessários para que se consiga controlar posição, velocidade e direcção do motor. Em termos genéricos, pode-se dividir a unidade de controlo em três sistemas: sistema de processamento, sistema de leitura de posição e sistema de comando de motor. Esta é uma divisão funcional, na medida em que a categorização dos sistemas se baseia na função que cada parte desempenha no conjunto da unidade de controlo. De seguida, é apresentado um diagrama funcional [Figura 35] que sintetiza a unidade de controlo.

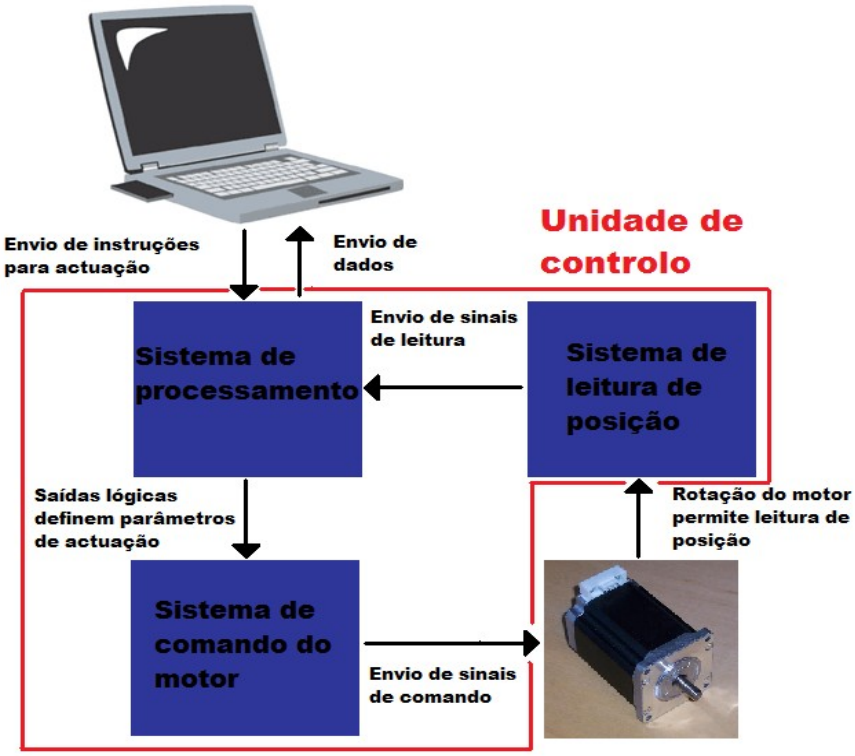


Figura 35: Diagrama funcional da unidade de controlo

Sistema de processamento

Este sistema é constituído por microcontrolador, divisor de frequência e porta série. O dispositivo central deste sistema é o microcontrolador [A12], sendo este responsável por:

- Gerar pulsos para controlo de velocidade do motor
- Controlar saídas lógicas que determinam a actuação do motor
- Receber e enviar mensagens para o computador remoto
- Registar os sinais provenientes do codificador

Todas as tarefas anteriormente descritas envolvem aquisição e processamento. O microcontrolador apenas actua em função das variáveis de entrada, sendo que essas podem ser de duas origens, do codificador ou do computador remoto. O codificador funciona como realimentação na medida em que dá informação ao PIC da real resposta do motor sobre o qual o PIC actua, ao passo que o computador remoto funciona como sinal de entrada na medida em que define a actuação sem ter em conta a resposta do motor. É no algoritmo interno que está previsto como actuar perante o estado das variáveis de entrada, sendo que actuação será sempre sobre três variáveis: direcção, velocidade e posição. O PIC também envia dados em função de variáveis de entrada, no entanto, não se considera esta uma variável de actuação.

A porta série desempenha essencialmente a função de recepção e envio de mensagens, sendo que é ela a interface de comunicação entre microcontrolador e PC. O envio de mensagens é fundamental no sentido de possibilitar ao computador remoto receber a informação actualizada da posição e velocidade do motor, ao passo que a recepção serve para que o computador remoto possa dar instruções de actuação para o PIC.

O divisor de frequência [A14] diminui a frequência proveniente do PIC. Apesar do PIC possuir *pre-scalers* internos não é possível obter uma gama de frequências adequada ao controlo do motor através do módulo PWM; nesse sentido torna-se necessário recorrer a um dispositivo externo que receba o sinal e o divida para valores em conformidade com os requisitos do motor. Seria, no entanto, possível programar uma saída para que fornecesse pulsos com as frequências desejadas, mas tal implicaria uma maior complexidade do algoritmo.

Sistema de leitura de posição

Este sistema é composto essencialmente pelo codificador. Como já foi mencionado antes, optou-se pelo codificador incremental; tal escolha implica que só após rotação do motor seja possível determinar a posição do LRF. Estando o codificador solidariamente ligado ao veio de rotação que suporta o LRF, é possível saber a posição do mesmo, mesmo havendo falhas no sistema de transmissão ou perda de pulsos no motor. O codificador transmite um sinal em pulso, sendo que cada pulso representa um incremento angular.

Sistema de comando do motor

Este sistema é constituído pelos *drivers* para motores passo a passo bipolares, os circuitos integrados L297 e L298 [A13]. A sua função é actuar sobre o motor obedecendo a alguns parâmetros definidos pelo PIC. Através de um único sinal de entrada proveniente do PIC, mais concretamente do divisor de frequência, são gerados 4 sinais pulsados com o desfasamento e voltagem adequados ao motor em questão.

4.2. Microcontrolador

Neste trabalho optou-se pelos microcontroladores da Microchip pelo facto de estes se encontrarem já disponíveis no laboratório. Para este projecto escolheu-se um microcontrolador da familia PIC18Fxxxx. A figura [Figura 36] que se segue mostra um conjunto de PICs de 28 portas, entre os quais alguns da série PIC18Fxxxx. Através das cores é possível ter uma ideia das portas que cada modelo possui. Refira-se, no entanto, que actualmente os PICs da série 18F24xx já possuem portas que ainda não lhe estão atribuídas nesta imagem, como por exemplo, as portas de *interrupt* INT1 e INT2.

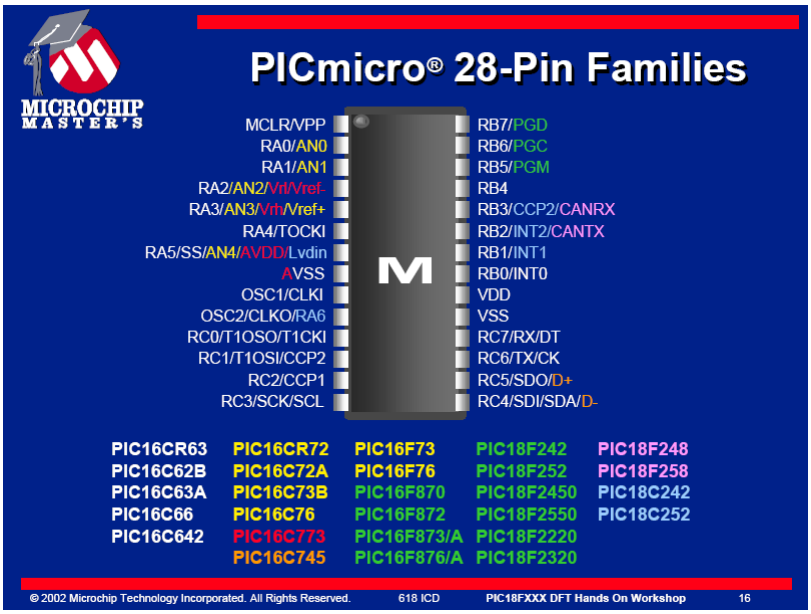


Figura 36: Microcontroladores de 28 pinos da Microchip (à data de 2002)

Entre os requisitos que estiveram na sua escolha pode-se enumerar os seguintes:

- Módulo PWM (CCP1)
- Duas entradas *Interrupt* (INT1 e INT2)
- Duas portas para comunicação série (RX e TX)
- Várias saídas e entradas analógicas e digitais

O modelo escolhido é o PIC18F2455, dado que preenche os requisitos e estava disponível no laboratório.

4.2.1. Configuração inicial do microcontrolador

Sempre que o PIC é reinicializado é preciso garantir que certos parâmetros estão definidos de acordo com os requisitos da aplicação, por isso mesmo, esses parâmetros são definidos inicialmente na rotina principal do programa.

A frequência do microcontrolador que define o tempo que dura um ciclo de instrução, ou seja o tempo que leva a ser executada uma instrução, pode ser de origem interna ou externa. Independentemente da origem da frequência, um ciclo de instrução [Figura 37] é executado a uma frequência quatro vezes inferior à frequência de relógio.

FIGURE 5-3: CLOCK/INSTRUCTION CYCLE

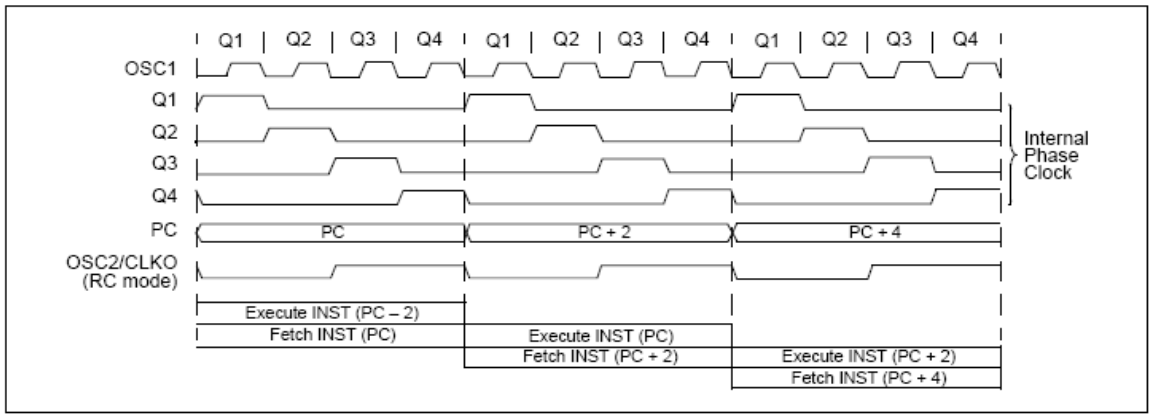


Figura 37: Ciclo de instrução

A frequência pode ser definida externamente por um oscilador de cristal e, recorrendo a *scalers* internos, é possível obter a diferentes frequências para diferentes tarefas. Para este projecto optou-se por um cristal de 20 MHz que permite ter um ciclo de instrução de 5 MHz.

Depois de escolhido o cristal, é possível configurar o *baud rate* [Figura 39] para o módulo de comunicação série. A definição do *baud rate* é feita através do registo SPBRG com o auxílio de um quadro de configuração [Figura 38], optando-se pela taxa de transmissão máxima de aproximadamente 115 Kbaud.

BAUD RATE (K)	SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRG16 = 1											
	Fosc = 40.000 MHz			Fosc = 20.000 MHz			Fosc = 10.000 MHz			Fosc = 8.000 MHz		
	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)	Actual Rate (K)	% Error	SPBRG value (decimal)
0.3	0.300	0.00	33332	0.300	0.00	16665	0.300	0.00	8332	300	-0.01	6665
1.2	1.200	0.00	8332	1.200	0.02	4165	1.200	0.02	2082	1200	-0.04	1665
2.4	2.400	0.02	4165	2.400	0.02	2082	2.402	0.06	1040	2400	-0.04	832
9.6	9.606	0.06	1040	9.596	-0.03	520	9.615	0.16	259	9615	-0.16	207
19.2	19.193	-0.03	520	19.231	0.16	259	19.231	0.16	129	19230	-0.16	103
57.6	57.803	0.35	172	57.471	-0.22	86	58.140	0.94	42	57142	0.79	34
115.2	114.943	-0.22	86	116.279	0.94	42	113.636	-1.36	21	117647	-2.12	16

Figura 38: Quadro de baud rates para modo assíncrono

```
//rs232 comunicação assincrona, envio de caracteres
TXSTAbits.TXEN=1;                //activar a escrita (Transmit enable)
TXSTAbits.SYNC=0;                //activar modo assincrono
BAUDCONbits.BRG16=1;  //activar 16 bits para o gerador de baud rate
SPBRG=42;                        //definição de baud rate de 116200
TXSTAbits.BRGH=1;                //selecção de modo baud rate elevado (high baud rate)
TXSTAbits.TX9=0;                 //negar nono bit, transmissão de 8 bits
RCSTAbits.SPEN=1;                //activar porta série (serial port enable)
RCSTAbits.CREN=1;                //activar recepção contínua
RCSTAbits.RX9=0;                 //negar nono bit, recepção de oito bits
```

```
RCONbits.IPEN=1;                //activação dos níveis de prioridade para interrupts
PIE1bits.RCIE=1;
```

Figura 39: Configuração dos parâmetros de comunicação RS232

As portas podem ser definidas como entradas ou saídas em função da tarefa que vão desempenhar. Os registos TRIS permitem definir a função de todas as portas de uma determinada categoria (A,B ou C), sendo que atribuindo o valor 1 ou 0 significa atribuir a uma porta a função de entrada ou saída respectivamente.

PCFG3:PCFG0: A/D Port Configuration Control bits:

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Figura 40: Quadro de configuração de portas analógicas e digitais

As portas do registo A foram todas configuradas como sendo digitais porque as portas que se pretendiam digitais eram as portas AN0, AN1e AN2, tal, como se pode ver pelo quadro [Figura 40], só era possível configurando todas as outras como digitais.

Os *timers* devem ser previamente configurados para permitir que na rotina de execução só seja necessário escrever no registo de execução TMRx. Neste caso será usado o timer0 [Figura 41] para temporizar o envio de mensagens; isto permite definir uma taxa de envio estável à medida das necessidades do projecto.

```
//definição do timer0 que irá temporizar o envio das mensagens
        T0CONbits.TMR0ON=1;         //activação do timer0
        T0CONbits.T0CS=0;           //definir o timer0 como um clock interno
```

```
T0CONbits.T08BIT=0;          //definir o contador para uma resolução de 16 bits
T0CONbits.PSA=0;             //activar prescaler para o timer0
T0CONbits.T0PS0=1;
T0CONbits.T0PS1=0;
T0CONbits.T0PS2=1;           //prescaler de timer 0 definido para 32
```

Figura 41: Configuração do timer0

É preciso definir os *interrupts* [Figura 42], em geral isso passa por activá-los, definir o seu nível de prioridade (baixa ou alta) e se ocorrem no instante em que se verifica uma descida (*falling edge)* ou elevação (*rising edge*) de tensão.

```
//definição de diversos interrupts
    INTCON3bits.INT2IP=1;        //nível de prioridade elevada para o interrupt INT2
    INTCON2bits.INTEDG2=0;       //interrupt "on falling edge"
    INTCON3bits.INT1IP=1;
    INTCON2bits.INTEDG1=0;
    IPR1bits.RCIP=0;             //baixa prioridade para o interrupt RCIF
    INTCON3bits.INT2IE=1;        //activação do interrupt INT2
    INTCON3bits.INT1IE=1;        //activação do interrupt INT1
    INTCON2bits.TMR0IP=1;        //definir alta prioridade para interrupt de TIMER 0
```

Figura 42: Configuração de interrupts

A configuração do módulo PWM deve atender aos requisitos de velocidade que se pretendam. Como já foi referido anteriormente, não se consegue obter a gama de frequências desejada directamente do módulo PWM, logo a configuração do sinal de saída deve ser conjugada com a divisão feita à saída. Para testar a exequibilidade dos parâmetros, simulam-se alguns valores no sentido de concluir se é possível obter a gama de frequências desejada. A fórmula para cálculo da frequência de saída é:

$$Freq = \frac{F_{osc}}{4 \times PR2 \times TMR2_{presc}}$$

O registo de oito bits PR2 é onde se insere o valor referente ao período do sinal, o $TMR2_{presc}$ é o *prescaler* do *timer* 2 que permite a divisão do sinal de frequência. O $TMR2_{presc}$ pode pode assumir o valor de 1, 4 ou 16 mediante a configuração do registo de dois bits T2CKPS1:T2CKPS0. F_{osc} é a frequência do oscilador de cristal, que no caso é 20 MHz. Se se adicionar à formula o divisor de frequência ($F_{divisor}$) tem-se:

$$Freq = \frac{F_{osc}}{4 \times PR2 \times TMR2_{presc} \times F_{divisor}}$$

Simule-se o valor máximo e mínimo para PR2, assumindo o valor unitário para $TMR2_{presc}$ e para o divisor de frequência.

$$\frac{20\text{x}10^6}{4\times1\times1\times1}=5\text{x}10^6\ Hz$$
$$\frac{20\text{x}10^6}{4\times255\times1\times1}=19608\ Hz$$

De seguida testa-se qual o valor mais adequado a adoptar no divisor de frequência assumindo novamente o valor unitário para $TMR2_{presc}$. A gama de valores do divisor de frequência oscila entre 2 (2^1) e 4096 (2^{12}) . Dado que as frequências obtidas são elevadas comece-se por tentar usar o máximo divisor.

Para 4096 (2^{12}):

$$\frac{5\text{x}10^6}{4096}\approx1220\ Hz$$
$$\frac{19608}{4096}\approx5\ Hz$$

Para 2048 (2^{11}):

$$\frac{5\text{x}10^6}{2048}\approx2441\ Hz$$
$$\frac{19608}{2048}\approx10\ Hz$$

Através das simulações é perceptível que o algoritmo não poderá usar toda a gama de valores possíveis no registo PR2, porque a frequência máxima excede a frequência máxima admissível pelo motor em regime de meio passo (doravante mencionada por F.M.A.M.P.), que é de 2000 Hertz. No entanto, fica também claro que é possível atingir a F.M.A.M.P com um $TMR2_{presc}$ unitário, o que é suficiente para se poder efectuar uma configuração inicial do módulo PWM. Outro factor a ter em conta na configuração é o factor de serviço (*Duty Cycle*). A definição do *Duty Cycle* é feita através do registo de 10 bits constituído pelo registo de 8 bits CCPR1L e os bits dos registos DC1B0:DC1B1, sendo que estes últimos são os bits menos significativos (LSB – Least Significant Bit). O mínimo período possível ocorre à máxima frequência ($F_{osc}/4=5\text{x}10^6$) logo será:

$$T_{min}=\frac{1}{5\text{x}10^6}=2\text{x}10^{-7}\ segundos$$

O *duty cycle* pode ser calculado pela seguinte expressão:

$$PWM_{DutyCycle}=[CCPR1L:CCP1CON(5:4)]\times T_{osc}\times(TMR2_{presc}\,Value)\Leftrightarrow$$
$$\Leftrightarrow[CCPR1L:CCP1CON(5:4)]=\frac{PWM_{DutyCycle}\times F_{osc}}{(TMR2_{presc}\,Value)}$$

O *duty cycle* terá de ser inferior a:

$$\frac{2\text{x}10^{-7}\times20\text{x}10^6}{1}=4=100_{bin}$$

Apesar de o *duty cycle* [Figura 43] para a frequência máxima ter de ser inferior ao calculado, pode-se assumir esse mesmo valor já que o algoritmo não irá utilizar a frequência máxima no controlo do motor.

```
//definição dos parâmetros para para PWM
        CCP1CONbits.DC1B1=0;        //os bits DC1B0:DC1B1 são os 2 LSBs
        CCP1CONbits.DC1B0=0;        //do registo que permite definir o duty cycle
        CCPR1L=1;                   //oito bits mais significativos (MSBs)
        TRISCbits.TRISC2=0;         //definir o resgisto RC2/CCP1/P1A como saída
        T2CONbits.TMR2ON=1;         //activar o timer 2
        T2CONbits.T2CKPS1=0x0;      //prescaler unitário
        T2CONbits.T2CKPS0=0x0;
```

Figura 43: Configuração do módulo PWM

4.2.2. Programação

Nesta secção será abordada a programação do microcontrolador, não de forma intensiva mas de forma modular, para explicar o objectivo das funções criadas.

O ambiente de desenvolvimento usado foi o do MPLAB disponível pelo fabricante [9].

Controlo de posição e velocidade

Saber a cada instante a posição do LRF é talvez a tarefa primordial deste sistema. É, afinal, da conjugação de coordenadas da posição do LRF e do *scan* levado a cabo pelo mesmo que resulta a possibilidade de obter dados tridimensionais do espaço envolvente. Primeiro, é preciso identificar os sinais provenientes do codificador quanto à sua tipologia e o que representam. Os sinais do codificador são sinais de pulso, sendo que cada pulso representa um incremento posicional [Figura 44]. Como já referido anteriormente, o codificador em questão tem uma resolução de aproximadamente 0.35º por pulso, logo cada pulso gerado pelo canal A (poderia-se também optar pelo canal B) deve ser interpretado como um incremento ou decremento dependendo do sentido de rotação.

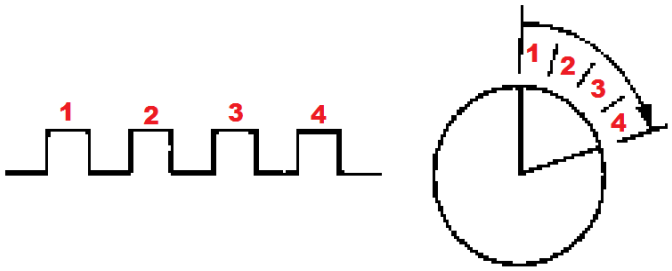


Figura 44: Sinal de pulso do codificador

Para determinar o sentido de rotação deve-se verificar o estado dos dois canais no

momento em que um deles muda de estado. Determinar a direcção do movimento [Figura 46] é possível pelo facto de os sinais estarem em quadratura [Figura 45]; estar em quadratura significa que os sinais estão desfasados em 90º. O desfasamento dos sinais permite identificar a direcção do movimento verificando o estado de um sinal quando o do outro muda.

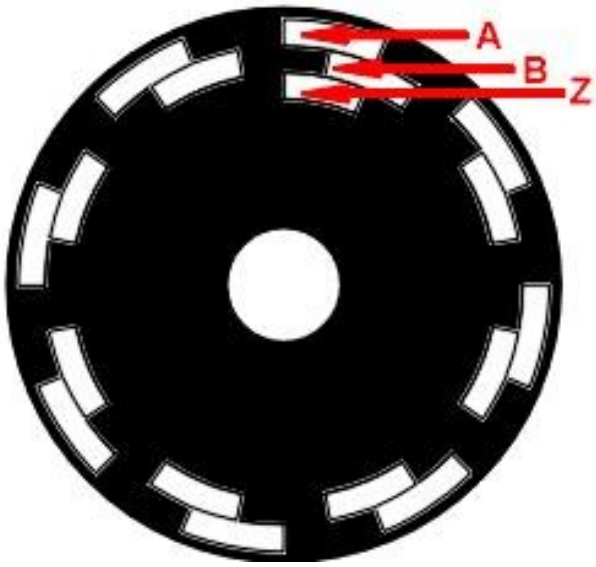


Figura 45: Disco interno de um codificador incremental

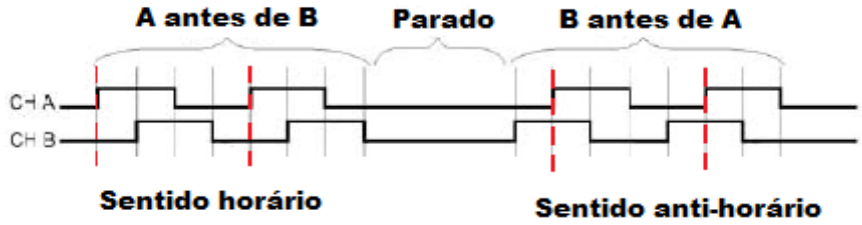


Figura 46: Interpretação dos sinais (Canal A e B)

Como se pode ver pelas figuras anteriores, se o movimento ocorrer no sentido horário, no exacto momento em que o canal A ficar activo (*rising level)* o canal B está inactivo, ao passo que se o movimento for anti-horário, no exacto momento em que o canal A ficar activo o canal B já está activo.

```
if (INTCON3bits.INT2IF==1) {                    //interrupt gerado pelo CANAL A (codificador)
        if (PORTBbits.RB3==1 && pos_ref==1) {           //se RB3 (CANAL B) já estiver activo
                cw=0;                           //...então o LRF move-se no sentido anti-horário
        }else if (PORTBbits.RB3==0 && pos_ref==1) {     //se RB3 (CANAL B) ainda estiver inactivo
                cw=1;                                   //...então o LRF move-se no sentido horário
        }
        if (cw==0 && pos_ref==1) {              //se o sentido de rotação for anti horário...
                deg1++;                         //...realizar 1 incremento posicional
        }else if (cw==1 && pos_ref==1) {        //se o sentido de rotação for horário...
                deg1--;                         //...realizar 1 decremento posicional
```

```
        }
        if ((deg1<0 || deg1>1024) && pos_ref==1) {          //se os incrementos forem superiores a 1024
                pos_ref=0;                                  //...então perdeu-se a posição de referência
                SendPointer(Wrong_pos);
        }
        deg=deg1*35;                                //cálculo da posição angular
        deg/=10;
        if (deg1==Pos_stop && pos_ref==1){          //se a posição actual coincidir com a (...)
                PWM_off();                          //...desactivar PWM
                send(deg,'NewLine');
                Pos_stop=4000;
        }
        pic_vel=5E6;                        //cálculo da velocidade definida no registo PR2
        pic_vel/=853;
        pic_vel/=PR2;

        INTCON3bits.INT2IF=0;               //o interrupt tem de ser "apagado" por software
}
```

Figura 47: Rotina de interrupt do canal A

Após definida a abordagem [Figura 47] sobre como interpretar os sinais provenientes dos canais A e B, resta saber como utilizar o canal 0 como complemento no processo de controlo de posição. O canal zero apenas emite um sinal por revolução, e é essa unicidade que o torna a base de referência neste processo. O facto de este canal só emitir um só sinal por revolução indica que se chegou a uma determinada posição que se assume como a posição zero.

O canal 0 também se revela um instrumento útil no controlo da velocidade. Através da contagem de *interrupts* gerados pelo timer0 é possível saber o intervalo de tempo entre cada *interrupt* do canal0. Sabe-se que cada *interrupt* do canal zero [Figura 48] representa uma revolução completa e que cada incremento do timer0 representa 13 milissegundos. Tendo noção do espaço e do intervalo de tempo que demora a percorrer esse espaço é possível obter a velocidade.

```
if (INTCON3bits.INT1IF==1) {                //interrupt gerado pelo CANAL 0 (codificador)
        k*=217;                             //cálculo da velocidade
        vel=1E6;
        vel/=k;
        k=0;
        if (cw==0) {                        //se o sentido de rotação for anti-horário...
                deg1=0;                     //...levar os incrementos do codificador a 0
        }else if (cw==1) {                  //se o sentido de rotação for horário...
                deg1=1024;                  //...levar os incrementos do codificador até 1024
        }
        if(pos_ref==1 && rs==1) {
                send(vel,'SameLine');
        }
        if(pos_ref==0 && RX==0) {
                pos_ref=1;                  //obtida posição de referência
        }
```

```
        INTCON3bits.INT1IF=0;           //interrupt "apagado" por software
}
```

Figura 48: Rotina de interrupt do canal 0

Em termos metodológicos, o processo de controlo de posição pode ser representado pelo seguinte diagrama [Figura 60]:

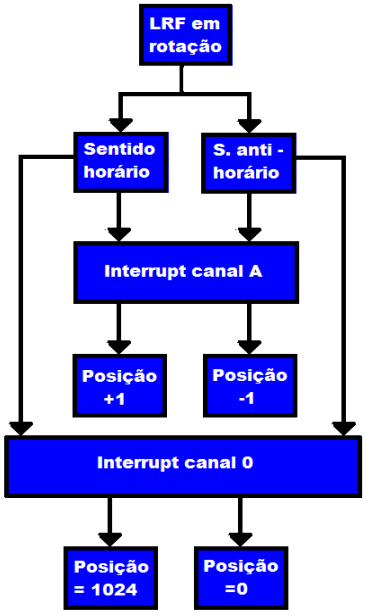


Figura 49: Diagrama do processo de controlo de posição

A posição armazena uma contagem linear dos pulsos, sendo que internamente o algoritmo multiplicará esse valor pela resolução angular obtendo assim a posição angular. O valor calculado é transmitido pela linha série com 4 algarismos, sendo o último algarismo um décimo de grau (o menos significativo). Uma das limitações do codificador incremental prende-se com a sua dependência do canal 0, dado que o controlo de posição só é iniciado após primeira detecção desse mesmo canal. A cada volta completada, o canal 0 serve como referência para garantir que a perda de pulsos não se "propague" influenciando o resto das medições; no entanto, entre cada volta não é possível garantir que não existem posições erradas. Tal sucede porque não é possível identificar o erro na sua génese mas sim pelas suas consequências:

- Se no intervalo de uma volta a posição for inferior a 0 ou exceder os 1024 pulsos;
- Se no instante em que se completa mais uma volta o valor da posição for menor ou maior que 1024 ou menor ou maior que 0;

No entanto, esta metodologia não permite identificar a posição angular em que isso ocorreu, nem tão pouco permite actuar no exacto momento em que ocorre. Apesar disso, o algoritmo actua assim que detecta a falha interrompendo o envio da posição angular pela linha série até que seja detectado novamente o canal 0.

Posição de referência ("*Home Position*")

Dado que o codificador é incremental, existe a necessidade de ter uma referência para que depois se saiba a posição em relação a essa referência. Nos codificadores incrementais essa referência é dada pelo canal zero; só após a primeira detecção desse canal é possível começar a enviar a posição pela linha série.

Sempre que o microcontrolador é inicializado, reinicializado ou o utilizador o especifique, é realizada a função *HomePos()* [Figura 50]. Esta função activa o sinal PWM a uma frequência baixa e pré-definida até que seja detectado o canal zero, assim que isso ocorrer o sinal é desligado e a variável que armazena a posição é reinicializada. Em termos operacionais o que se sucede é que o LRF roda a uma velocidade baixa até ao instante em que é detectado o canal zero, sendo logo imobilizado [Figura 51]. Esta função foi concebida sem que fosse gerado um *interrupt,* tendo se optado por verificar o estado da porta num ciclo *while()* e desactivando o sinal PWM assim que ela mudasse de estado. A diferença reside no facto de o *interrupt* interromper a rotina que está a ser executada e executar a rotina associada ao mesmo, ao passo que, neste caso, a rotina é executada e só passa para a instrução seguinte quando a condição se verificar. Isto permite poupar código escrito na rotina do *interrupt* e evita a criação de novas variáveis. Note-se que o *interrupt* do canal zero terá mais propósitos para além de auxiliar a função *Home Position()*, logo seria necessário criar mais variáveis que permitissem saber que propósito servia o *interrupt* em determinado momento.

```
int HomePos() {
        INTCON3bits.INT1IE=0;
        INTCON3bits.INT2IE=0;
        PWM_on();
        while(INTCON3bits.INT1IF==0);
        PWM_off();
        INTCON3bits.INT1IF=0;
        INTCON3bits.INT1IE=1;
        INTCON3bits.INT2IE=1;
}
```

Figura 50: Função HomePos()

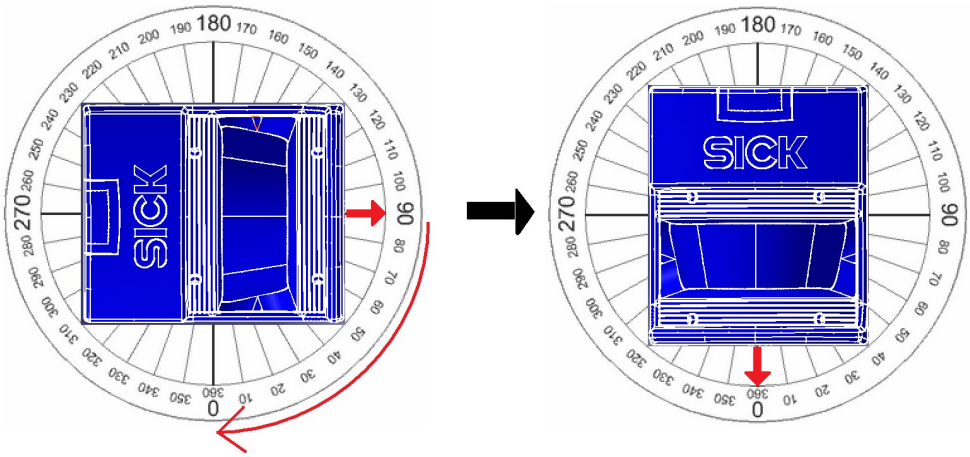


Figura 51: Movimento do LRF na procura da posição de referência

Rampa de aceleração

Dado que o LRF apresenta alguma inércia relevante, tanto o arranque como as paragens devem ser suaves. A melhor maneira de garantir isso é criando uma rampa de aceleração que garanta que não haja transições bruscas de velocidade.

A função *SpeedRamp()* [Figura 52] basicamente recorre a duas variáveis, o período actual e o período final do sinal que se pretende alcançar. A primeira tarefa consiste em saber se o período actual é zero, pois tal implica uma abordagem diferente no algoritmo. Também é necessário comparar os períodos para perceber se se terá de criar uma rampa de aceleração ou desaceleração.

O período do sinal é determinado através do registo de oito bits PR2; dado que este também é legível, é possível saber a frequência actual lendo esse mesmo registo. Um registo de oito bits permite uma gama de valores de 0 a 255; ora em PR2 determina-se o período do sinal, logo quanto menor o período maior a frequência. Para criar uma rampa de aceleração é necessário diminuir os valores do registo PR2 e para criar uma rampa de desaceleração é preciso aumentar os valores de PR2. Para fazer uma rampa de aceleração a partir de zero não é possível usar a leitura directa do registo PR2 como ponto de partida. Tal sucede porque não se pode decrementar a partir do valor 0, sendo que a frequência mais baixa está associada ao valor mais alto do registo PR2. Logo, caso o registo PR2 esteja a zero, o algoritmo deve atribuir um valor elevado para o registo PR2 e fazer a rampa de aceleração decrementando a partir desse valor.

Foi decidido que o motor funcionará em regime de meio passo (*Half step)* com passos de 0,9º. Esta opção prende-se com três factores:

- Maior resolução para posicionamento;
- Funcionamento mais suave do que no regime de passo completo*;*

A opção pelo regime de meio passo implica que, para a mesma velocidade, se tenha de ter o dobro da frequência, sendo que a frequência máxima ronda os 2000 Hz (embora experimentalmente se tenha verificado que consegue dar resposta a frequências um pouco mais elevadas). Tal compreende-se pelo facto de cada pulso gerado em regime de meio passo ter metade da amplitude angular do que em regime de passo completo.

Como as frequências mais elevadas são conseguidas com os valores mais baixos do registo PR2, verifica-se que entre valores sucessivos existem rácios entre frequências muito elevados, sendo o máximo uma relação incremental de 1 para 2 ou decremental de 2 para 1. Este aspecto diminui a resolução para as frequências mais elevadas e isso, por sua vez, compromete a qualidade da rampa. Ressalve-se que alterar a relação de transmissão motor-LRF nada alteraria nesse sentido. Embora reduzisse, em termos absolutos, o valor dos incrementos ou decrementos isso resultaria também na redução da velocidade final, sendo que a progressão geométrica da velocidade se manteria inalterada. Uma alternativa para conseguir boas resoluções para todos os valores passaria por mudar o $TMR2_{presc}$ em função da frequência, tal implicaria que, durante a rampa de aceleração. Esta solução poderá ficar para futuros trabalhos. Optou-se por ter um divisor de frequência menor (128)

e um $TMR2_{presc}$ fixo unitário, tal resulta num compromisso entre obter uma gama de frequências mais elevadas do que o necessário mas ter uma melhor resolução para as frequências perto da F.M.A.M.P.. Isto requer algum cuidado no desenvolvimento do algoritmo, já que PR2 permite obter frequências acima da F.M.A.M.P. que seriam prejudiciais para o motor.

```
void SpeedRamp(int Pres,int OldPres){                    //rampa de velocidades
        unsigned int rp,fraq;

        if (PORTAbits.RA0==0) {
        PORTAbits.RA0=1;
        }
        if (OldPres==0) {
        OldPres=245;
        PWM_on();
        }
        rp=OldPres;

        if (Pres>OldPres) {                              //Rampa de aceleração
        while (rp<Pres+1) {
                if (rp>150) {
                        wait(5);
                }else if (rp>50 && rp<=150){
                        wait(10);
                }else if (rp<=50){
                        wait(20);
                }
                rp++;
                PR2=rp;
        }
        }else if (Pres<OldPres){                         //Rampa de desaceleração
        while(rp>Pres-1) {
                        if (rp>150) {
                        wait(5);
                }else if (rp>50 && rp<=150){
                        wait(10);
                }else if (rp<=50){
                        wait(20);
                }
                rp--;
                PR2=rp;
        }
        }else if (Pres==OldPres) {
        PR2=Pres;
        }
}
```

Figura 52: Função SpeedRamp()

Comunicação série

O envio de caracteres pela linha série envolve um processamento anterior, isto

porque a natureza dos dados assim o exige. Neste projecto existem três funções para envio de dados pela linha série: SendArray(), SendPointer() e send().

A função SendArray() [Figura 53] tem duas variáveis de entrada, o *array* e a sua dimensão. O processo de envio consiste em enviar carácter a carácter pela linha série, isto é feito enviando o conteúdo associado a cada indice do *array* até que se tenha atingido o fim do mesmo.

```
void SendArray(char array[],int dim) {          //envio de arrays
        int s;
        for (s=0;s<dim;s++) {
                while (PIR1bits.TXIF==0);
                TXREG=array[s];
        }
}
```

Figura 53: Função SendArray()

A função SendPointer() [Figura 54] envia conteúdos de ponteiros. O processo é semelhante ao envio de *arrays,* enviando carácter a carácter, no entanto, o processo é terminado quando se detecta o carácter 0.

```
void SendPointer(rom near char *nline) {          //envio de ponteiros

        while (*nline!=0) {
        while(PIR1bits.TXIF==0);
        TXREG=*nline++;
        }
}
```

Figura 54: Função SendPointer()

A função send() [Figura 55] é a função mais complexa das funções de envio de caracteres, isto porque implica a conversão de decimal em ascii. As variáveis de entrada são o valor a ser transmitido e uma string indicando se o envio deve ser feito na mesma linha ou na linha seguinte. Esta função converte o valor em ascii e armazena-o num *array*, este, por sua vez, é enviado através da função SendArray().

```
void send(int count,char NL) {

        unsigned int rest,lap1,lap2;
        int v=0,u,w;
        static char str[];

        if (NL=='NewLine') {
                while(PIR1bits.TXIF==0);
                TXREG=0xa;
        }else if (NL=='SameLine') {
                while(PIR1bits.TXIF==0);
                TXREG=0x20;
        }
```

```
        while (lap1>0) {            //contabiliza-se quantos algarismos tem o número
                v++;
                lap1/=10;
        }
        for (u=v-1;u>0;u--) {       //contabilizados os algarismos, inseri-los num array
                rest=lap2%10;
                rest+=0x30;
                str[u]=rest;
                lap2/=10;
        }
        str[0]=lap2+0x30;

        SendArray(str,v);           //enviar array por linha série
}
```

Figura 55: Função send()

A recepção de caracteres é feita na rotina de *interrupt* do buffer de recepção RCIF [Figura 56]. Os caracteres são armazenados num *array* até que seja detectado o carácter indicativo de fim de mensagem, posteriormente o *array* será analisado na rotina principal para que se possa executar as instruções de comando.

```
#pragma interruptlow low_isr

void low_isr (void) {                       //rotina para interrupts de baixa prioridade

        If (PIR1bits.RCIF==1) {             //interrupt gerado pela recepção de dados via RS232
                i=0;
                while (RCREG!='\r') {       //"enquanto" não chegar o caracter de fim de mensagem...
                        str2[i]=RCREG;      //...os caracteres são armazenados no array str2
                        while(PIR1bits.RCIF==0);  //esperar pelo interrupt gerado pelo caracter
                        i++;
                }
                RX=1;               //variável que indica que chegou uma mensagem pela linha série
                str2[i]='\r';       //sinalização do fim do array
        }
}
```

Figura 56: Rotina de interrupt para recepção de mensagem

Determinou-se que este *interrupt* seria de baixa prioridade dado que no topo das prioridades estavam os *interrupts* responsáveis por receber os sinais do codificador.

Imobilização do motor

Durante a operação, o utilizador pode desejar parar o motor. Para esta aplicação criou-se a imobilização com bloqueio temporário, embora o motor não fique permanentemente bloqueado na mesma posição, este é bloqueado durante um breve período de tempo para garantir que não continua em rotação devido à sua inércia. Esta opção prende-se com o facto de a imobilização com bloqueio consumir muita corrente. A

imobilização é feita desactivando o sinal PWM (CCP1M1:CCP1M3=0000) e o desbloqueio desactivando a porta RA0 que liga ao circuito integrado L297. É preciso ter sempre presente a inércia do LRF, logo a imobilização será precedida de uma rampa de desaceleração de forma a garantir que é feita a uma velocidade reduzida sem danos para o sistema.

A imobilização também pode ser feita com controlo de posição, o utilizador pode requerer que o LRF seja imobilizado numa determinada posição. A possibilidade de determinar a posição de paragem do LRF apresenta-se como uma boa ferramenta de despistagem de erros porque, com o auxílio de um circulo com coordenadas angulares, é possível verificar visualmente a menor ou maior correcção do posicionamento. A imobilização com posicionamento [Figura 57] é realizada na rotina de *interrupt* do canal A porque é aí que é actualizada a posição do LRF. Assim que a posição actual for igual à posição de comando enviada pelo utilizador o motor é imobilizado.

```
if (deg1==Pos_stop && pos_ref==1){        //se a posição actual coincidir com a posição de paragem
        PWM_off();                        //...desactivar PWM
        send(deg,'NewLine');
        Pos_stop=4000;
}
```

Figura 57: Algoritmo para imobilização com posicionamento

Mudança de direcção

O utilizador tem também a possibilidade de mudar o sentido de rotação [Figura 58] do LRF através da linha série.

As leis da Física determinam que um corpo que mude de direcção tem de obrigatoriamente passar pelo "repouso", ou seja, no preciso instante em que se dá a mudança de direcção o corpo tem de estar parado. Se o LRF estiver em movimento, a mudança de direcção é feita criando uma rampa de desaceleração e imobilizando-o; de seguida é alterado o estado da porta RA1 que define o sentido de rotação e é criada uma rampa de aceleração até à velocidade inicial. Caso o LRF já se encontre imobilizado, então apenas se muda o estado da porta RA1 de forma a que, quando for dada ordem de movimento, este comece a rodar no sentido definido pelo utilizador.

```
}else if (str2[j]=='f' || str2[j]=='F') {    //movimento sentido horário
        if (PORTAbits.RA1==1 && PORTAbits.RA0==1) {
                Speed=PR2;
                SpeedRamp(245,PR2);
                PWM_off();
                PORTAbits.RA1=0;
                wait(1000000);
                SpeedRamp(Speed,PR2);
        }else if (PORTAbits.RA1==1 && PORTAbits.RA0==0) {
                PORTAbits.RA1=0;
```

```
}else if (PORTAbits.RA1==0) {
        SendPointer(Invalid_rot);
}
```

Figura 58: Algoritmo para rotação em sentido horário

4.2.3. Protocolo de comunicação

Este projecto baseia-se na possibilidade de um utilizador poder determinar algumas acções através de um computador remoto, sendo a comunicação levada a cabo via RS232. Para que tal seja possível, é necessário criar uma linguagem de baixo nível, um protocolo de comunicação, onde fique pré-estabelecido que caracteres correspondem a que acções.

Estrutura da mensagem

A definição da estrutura da mensagem é vital para que esta possa ser interpretada correctamente. O microcontrolador deverá esperar uma estrutura lógica condicionada, dado que o algoritmo não pode estar preparado para receber uma infinidade de estruturas diversificadas. Uma das primeiras condições a estabelecer é o terminador da mensagem, ou seja, o carácter que será interpretado pelo algoritmo como sendo o fim da sequência de caracteres enviada. A decisão deverá passar por escolher um carácter que não seja necessário como carácter da própria mensagem em si; por isso se opta normalmente por caracteres que representam acções de escrita e não caracteres de escrita em si. Escolheu-se o carácter "\r" (carriage return) como terminador de mensagem.

Também seria possível definir uma dimensão rígida para a mensagem, estipulando, por exemplo, uma mensagem de 2 bytes, onde a cada um dos 16 bits estivesse associada uma instrução específica. No entanto, no âmbito deste projecto optou-se por algo manifestamente diferente; primeiro determinou-se que a lógica de comando seria baseado numa lógica de carácter(s)-comando em vez de bit(s)-comando, ou seja, a análise da mensagem será feita carácter a carácter e não bit a bit. Embora isto implique uma maior quantidade de dados, tem como contrapartida o facto de ser mais acessível e intuitivo para o utilizador (*"user-friendly"*). Será mais fácil ao utilizador associar letras a determinadas acções do que associar bits em determinadas posições a determinadas acções. Isto permite que o próprio controlo directo por linha série (HyperTerninal, Docklight, Cutecom...) seja muito mais fácil, sendo mais intuitivo para o utilizador e tornando a ausência de uma interface gráfica própria no computador remoto menos problemática. Também a sequência da mensagem não é rígida; isto é possível precisamente por este protocolo estar baseado numa lógica de carácter(s)-comando.

O facto da sequência da mensagem não ser rígida tem o propósito de possibilitar que o utilizador defina a ordem de execução [Figura 59] das acções ordenando as mesmas no *array*. Depois, basta ao algoritmo processar as ordens lendo o *array* do início para o fim e executar as acções à medida que as identifica.

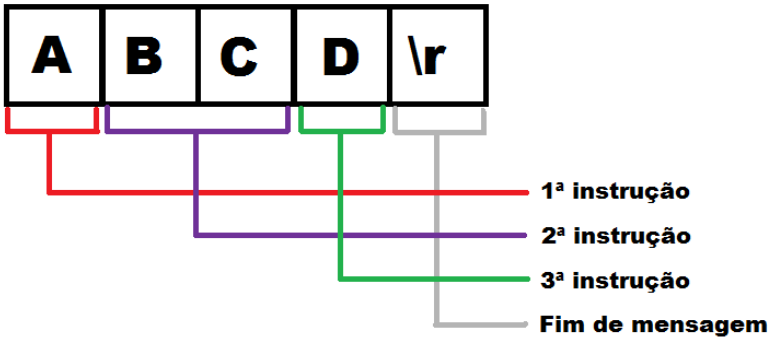


Figura 59: Sequência de execução de mensagem

Também a dimensão da mensagem não é rígida, e embora na prática esta tenha limites, o utilizador pode enviar mensagens de dimensão variável. O limite da mensagem prende-se com o facto de não serem aceites comandos repetidos, portanto o limite máximo estará definido pela diversidade de comandos possíveis.

Mensagem de instruções

Como já foi referido, cada mensagem pode ser constituída por um conjunto de instruções. Todas as instruções são aceites em maiúsculas ou minúsculas.

H → "Homeposition"

O carácter "h" indica que se deverá imobilizar o LRF na posição de referência. Como este comando pode ser enviado em pleno movimento, a acção é precedida de uma rampa de desaceleração até à velocidade mais baixa, só atingida esse velocidade é que se procede à imobilização do LRF na posição zero.

S_ _ _ → "Speed"

O carácter "s" deve ser utilizado para definir a velocidade de rotação do LRF. Atente-se que o algoritmo interpreta a velocidade inserida com sendo a velocidade que o utilizador pretende que o LRF rode e não como a velocidade de rotação no veio do motor, ou seja, em termos internos este terá em conta o factor relação de transmissão. A seguir ao carácter "s" deve estar o valor, em ascii, da velocidade pretendida em rotações por minuto. Como a rotação máxima do LRF é de 300 r.p.m., o utilizador pode inserir até 3 algarismos a seguir ao carácter inicial. Caso o valor da velocidade seja composto por menos de 3 algarismos, o utilizador não precisa de preceder o valor de zeros para preencher todas as casas, pode enviar a velocidade somente com os algarismos que a constituem.

M → "Motion (disabled)"

Esta instrução serve para imobilizar o LRF sem controlo de posição. A imobilização é precedida da rampa de desaceleração, quando esta for terminada. O LRF é

imobilizado.

P_ _ _ _ → “Position”

A imobilização com controlo de posição é activada através do carácter “p” seguido da posição de paragem constituída, no máximo, por 4 algarismos. À semelhança da posição enviada pela linha série, a posição de comando deve ser explicitada até à casa das décimas de grau não sendo, contudo, obrigatório preencher as 4 casas de algarismos. O utilizador pode requerer que o LRF seja imobilizado numa posição à qual a resolução angular do motor, mesmo com a relação de transmissão, não consiga responder, por isso mesmo a posição à qual este é imobilizado é enviada por linha série.

F → “Forward”

Assumindo o sentido de horário como sendo “para a frente”, o mesmo pode ser conseguido enviando o carácter “f”. A opção por esta terminologia prende-se com o facto de os “convencionais” “cw” e “ccw” (*Clockwise* e *Counterclockwise* respectivamente) requerem mais caracteres.

Caso o LRF esteja em rotação e em sentido oposto ao desejado pelo utilizador, é criada uma rampa de desaceleração até à imobilização, sendo depois iniciada uma rampa de aceleração até à velocidade inicial mas em sentido contrário. Caso o LRF já esteja a rodar nesse sentido é enviada pela linha série uma mensagem notificando isso mesmo, se o LRF estiver imobilizado apenas é alterado o estado da porta que define o sentido.

B → “Backward”

O sentido de rotação anti-horário é convencionado como estando a rodar “para trás”. A instrução é em tudo semelhante à instrução “f”.

V_ → “Verbose”

Este comando serve para inibir ou activar o envio da posição do LRF via RS232. A escrita é inibida se o carácter for procedido de um 0 e activada se o carácter for procedido de um 1. Porém, a escrita pode também ser interrompida pelo algoritmo interno caso o mesmo detecta alguma anomalia no controlo de posição.

A título de exemplo, imagine-se que o utilizador enviava a seguinte array [Figura 60] de instruções:

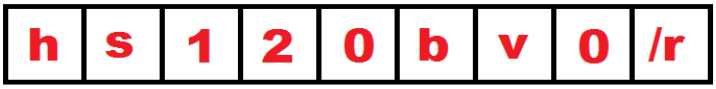


Figura 60: Array de instruções

A execução deste array é ilustrada pelo diagrama [Figura 61] que se segue.

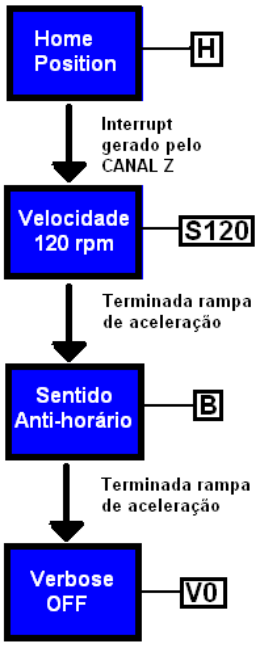


Figura 61: Diagrama para exemplo de instrução

Note-se que a ordem das instruções respeita a ordem das mesmas no corpo da mensagem.

Análise da mensagem

Antes da execução, a mensagem deve ser analisada no sentido de averiguar a sua validade lógica. A validação da mensagem passa por verificar as seguintes condições:

- Não existe nenhum carácter não reconhecido
- Não existem instruções repetidas
- Só existe uma instrução de direcção
- A seguir à instrução de velocidade encontram-se entre 1 a 3 algarismos
- A seguir à instrução de paragem com comando de posição existem entre 1 a 4 algarismos
- A seguir à instrução de envio por linha série existe um algarismo que é 1 ou 0

O não cumprimento das condições acima descritas implica que a mensagem seja descartada. A mensagem é desaproveitada porque se considera que a estrutura lógica da mensagem vale no seu todo, sendo que a ser validada em parte não respeitaria as indicações do utilizador.

As instruções repetidas não serão aceites, porque assume-se que não faz sentido na

mesma mensagem requerer dois estados para a mesma "variável". Por exemplo, não fará sentido estabelecer duas velocidades, isto porque a velocidade final de operação será só uma. Para o efeito, a mudança de estado sobre a mesma variável requer uma nova mensagem indicando isso mesmo.

Processamento e execução da mensagem

Se da análise da mensagem resultar que a mesma cumpre as condições lógicas acima mencionadas, então esta será válida e dar-se-á seguimento ao seu processamento e execução. O processamento da mensagem é feito identificando carácter a carácter até que seja reconhecida uma instrução, assim que isso aconteça a instrução é executada. No fim da execução da instrução o algoritmo procede com a averiguação dos restantes caracteres até atingir o fim da mensagem.

4.3. Unidade de potência

A unidade de potência é o agregado de todos os elementos físicos necessários para que o sinal pulsado proveniente do microcontrolador seja convertido num conjunto de sinais pulsados adaptados ao motor em questão. Dado que em trabalhos anteriores [6] os drivers para motores bipolares foram utilizados com sucesso, optou-se por utilizar o motor unipolar como sendo bipolar [Figura 62]. Existem seis fios de ligação ao motor sendo este constituído por dois enrolamentos, cada enrolamento é constituído por 3 fios sendo um deles o fio central. Para utilizar um motor unipolar de 6 fios como bipolar devem se deixar os dois fios centrais desligados, o fio preto(1) e branco(6).

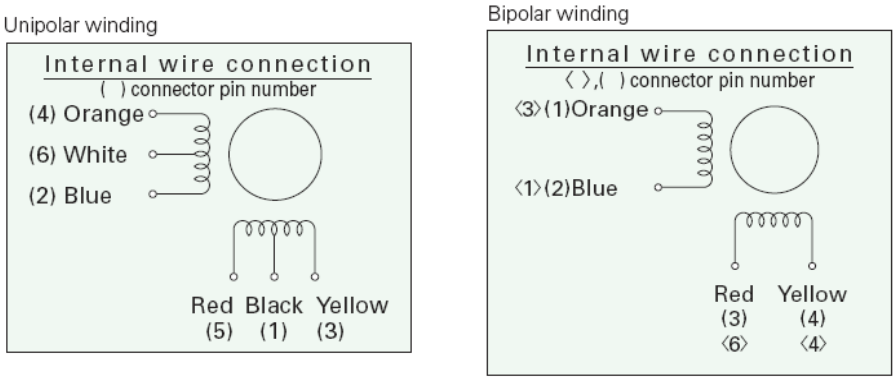
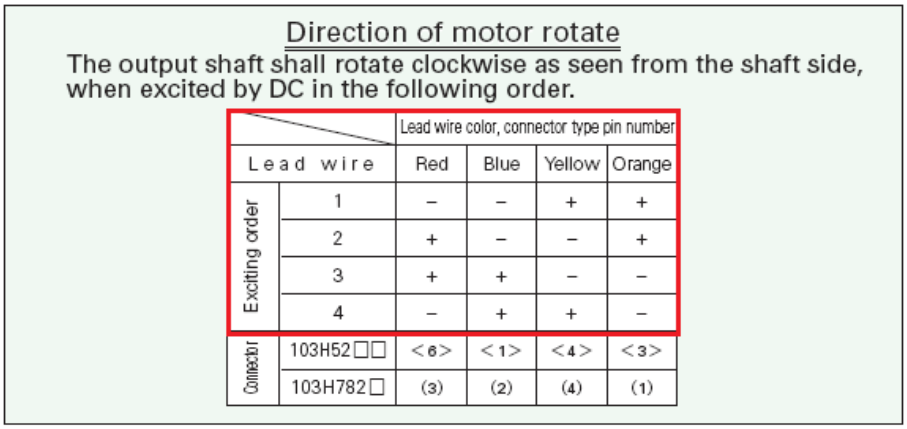


Figura 62: Esquema de ligações internas unipolar e bipolar

Sinais de controlo do motor

A primeira etapa consiste gerar quatro sinais através do sinal proveniente do microcontrolador, esses sinais devem ter o desfasamento adequado para alimentar as fases de forma sequencial. É através da activação sequencial das fases que é possível obter a

rotação no sentido horário ou anti-horário. A sequência dos sinais [Figura 63] é indicada pelo fabricante [11].



Direction of motor rotate

The output shaft shall rotate clockwise as seen from the shaft side, when excited by DC in the following order.

		Lead wire color, connector type pin number			
Lead wire		Red	Blue	Yellow	Orange
Exciting order	1	–	–	+	+
	2	+	–	–	+
	3	+	+	–	–
	4	–	+	+	–
Connector	103H52□□	<6>	<1>	<4>	<3>
	103H782□	(3)	(2)	(4)	(1)

Figura 63: Sequência de sinais para motor bipolar

Como é possível observar, cada fase pode alternar o sentido da corrente que a atravessa, isto com intuito de criar a cada instante um campo magnético favorável ao movimento, neste exemplo, em sentido horário. Se cada fase tem dois sentidos de corrente, daí resulta que existam quatro combinações possíveis de activação das fases. O processo acima descrito é cíclico, sendo que cada etapa do ciclo representa uma deslocação de passo completo de 1,8º.

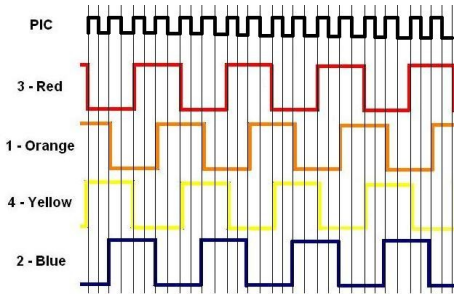


Figura 64: Sinais PWM em regime de passo completo

A figura acima [Figura 64] mostra a relação entre os sinais do motor e o sinal original do microcontrolador. Repare-se que o sinal de cada entrada tem uma frequência 4 vezes inferior à frequência do PIC; tal resulta do facto de as quatro entradas actuarem sequencialmente para produzirem o mesmo efeito que um único sinal PWM. O desfasamento entre sinais é de um período do sinal original, precisamente para que haja uma elevação de tensão com a mesma frequência que no sinal original.

Capítulo 5

Resultados e conclusões

Estrutura

A estrutura mostrou-se sólida e coesa, apesar de o protótipo produzir algum ruído. Rodando manualmente o LRF verifica-se que os níveis de ruído são imperceptíveis, sendo que o ruído provirá maioritariamente da trepidação transmitida pelo motor à estrutura. Elementos como o motor, o transformador, o LRF, o codificador, as baterias e o *slip ring* foram incorporados na estrutura como previsto. Para o *slip ring* foi necessário realizar duas furações especificamente orientadas para que os cabos de alimentação e comunicação chegassem à parte inferior da estrutura.

Controlo de direcção

O controlo de direcção revelou-se eficaz e bem complementado pelas rampas de aceleração, garantindo que a mudança de direcção era feita só quando o LRF estivesse imobilizado. No entanto, a averiguação do sentido de rotação como mecanismo auxiliar de contagem de pulsos revelou-se falível, principalmente para baixas frequências. Saber se o LRF rodava em sentido horário ou anti-horário [Figura 46] é fundamental para saber permanentemente a posição em relação ao referencial, como está patente no diagrama [Figura 49] já apresentado. A averiguação do sentido de rotação é feita cada vez que é alcançada uma nova posição, ou seja, cada vez que é gerado um *interrupt no* canal A. Se não houver perdas de pulsos e o sentido de rotação estiver sempre correcto e actualizado, será expectável que não haja erros na determinação da posição. No entanto, verificou-se que algumas vezes as posições enviadas excediam os 360º, por outras palavras, num período de uma volta completa o PIC recebeu mais pulsos do codificador do aqueles que ele possui por revolução. Uma explicação possível baseia-se em pequenas descontinuidades no movimento. Estando o LRF a rodar num sentido, este teria um pequeno recuo no sentido contrário ao movimento sem activar o pulso anterior, logo o PIC não receberia a informação que houve um recuo momentâneo. Ao receber novamente o pulso, este incrementaria uma posição apesar de estar a passar novamente pela mesma posição.

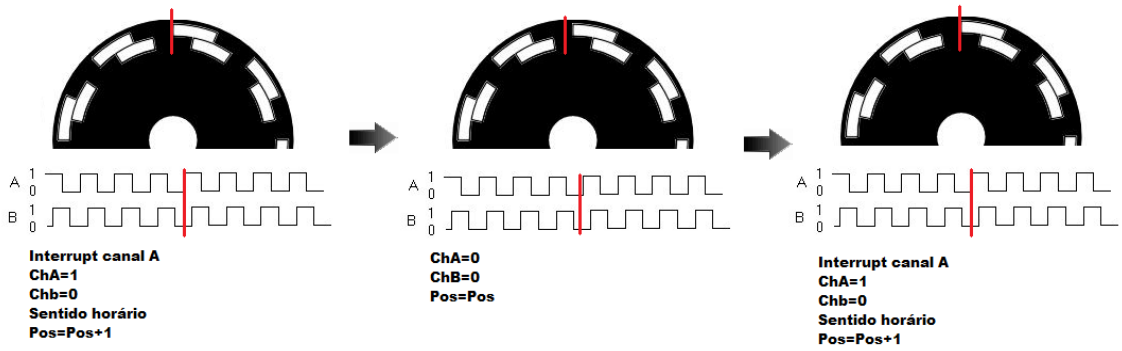


Figura 65: Erro de posição

Posicionamento

O excesso de velocidade pode levar a perda de pulsos e, consequentemente, gerar discrepância entre a posição real e a posição obtida através do microcontrolador. Dados do fabricante indicam uma taxa máxima pulsos de 160 kHz, ora, a velocidade máxima é de 300 r.p.m. e o codificador tem 1024 P.P.R..

$$Frequência_{máxima} = \frac{300 \times 1024}{60} = 5120\ Hz$$

A máxima taxa de pulsos admissível pelo codificador está bastante sobredimensionada em relação à frequência máxima que pode resultar deste projecto, pelo que a perda de pulsos por via do codificador é improvável. Mais improvável ainda é perda por software, visto que o microcontrolador funciona com frequências na casa dos MegaHertz.

Para averiguar a maior ou menor correcção do posicionamento, utilizou-se um circulo com coordenadas angulares. Esse mecanismo de despistagem permitiu perceber que efectivamente a posição que era enviada pela linha série estava dentro de uma margem de erro de 1º. A margem na realidade pode ser menor, mas dois factores diminuem a capacidade de despistagem de erros:

- A dificuldade em garantir que a posição zero do codificador está alinhada com a posição zero do círculo de coordenadas angulares
- A resolução do círculo de coordenadas ser de 1º

Este é um mecanismos de despistagem grosseiro, relembre-se que a resolução do codificador ronda os 0,35º; o que significa que se está a avaliar a performance do codificador com uma margem de erro de 3 a 4 pulsos. No entanto, permite afirmar com clareza que, em rotação contínua no mesmo sentido, o erro entre posição posição obtida pelo codificador e posição real não excede 1º.

Não se tendo criado uma metodologia mais rigorosa para averiguar disparidade entre valores reais e valores obtidos através do codificador, pode-se contudo verificar, por

exemplo, qual a disparidade entre a ordem de paragem enviada pelo utilizador e posição de paragem ordenada pelo microcontrolador. Essa disparidade está relacionada com o algoritmo e os seus métodos de cálculo. Como já foi mencionado, a ordem de comando vem em décimos de grau, mas como o utilizador pode requerer uma posição de paragem à qual o codificador não tenha resolução suficiente para responder, são realizados cálculos no sentido de encontrar a posição mais próxima da posição de comando. Os resultados [Tabela 1] são os seguintes:

Comando (º)	Retorno (º)
5,0	4,9
15,0	14,7
30,0	29,7
45,0	44,8
60,0	59,8
75,0	74,9
90,0	89,9
105,0	105,0
120,0	119,7
135,0	134,7
150,0	149,8
165,0	164,8
180,0	179,9
195,0	194,9
210,0	210,0
225,0	224,7
240,0	239,7
255,0	254,8
270,0	269,8
285,0	284,9
300,0	299,9
315,0	315,0
330,0	329,7
345,0	344,7

Tabela 1: Posição de comando vs posição retorno

Para esta amostragem o erro médio é de 0,18º e o erro máximo é de 0,3º, ou seja o erro não excede a resolução do codificador de 0,35º. Tal pode ser considerado como indicador de que o algoritmo está a processar correctamente a informação proveniente do computador remoto. A diferença entre a posição de comando e de retorno prende-se com a resolução do codificador. Apesar de o utilizador poder requerer posições com resolução de 0,1º, a resolução do codificador é de 0,35º, sendo que o máximo afastamento admissível

entre posição de comando e de retorno deverá ser a resolução do codificador. Caso contrário, significaria que a disparidade entre posições resultaria de erros de cálculo e não de limitações materiais concretas, já que o codificador permitiria obter uma posição mais próxima da posição de comando. De notar que o valor da posição é um inteiro constituído por quatro algarismos que resulta do arredondamento de um valor fraccionário com 5 algarismos (inclui centésimas de grau). O arredondamento é feito sempre por defeito, isto porque armazenando o resultado como inteiro este ignora a parte fraccionária do valor.

Velocidade

A velocidade do motor é controlada através do controlo da frequência dos pulsos emitidos para o motor. Para averiguar se a velocidade pretendida está a ser correctamente transmitida ao motor é necessário averiguar, através do osciloscópio, se a frequência transmitida está correcta [Tabela 2].

Velocidade de comando (rpm)	Frequência osciloscópio (Hz)	Velocidade*
30	200	30
45	300	45
60	400	60
75	500	75
90	600	90
105	710	106,5
120	810	121,5
135	900	135
150	1000	150

Tabela 2: Velocidade de comando vs frequência de sinal de pulso

*A velocidade final é igual a:

$$Velocidade = \frac{Frequência_{osciloscópio} \times 0{,}9 \times 60}{360}$$

De referir que a resolução do osciloscópio é de 10 Hz, logo existe um intervalo de tolerância de ±1,5 r.p.m..

Uma das formas de averiguar o controlo da velocidade é através da posição controlada no tempo, sabendo o intervalo de tempo entre cada transmissão de posição é possível averiguar a velocidade. Veja-se a seguinte amostra [Figura 66] para uma

velocidade de 50 r.p.m..

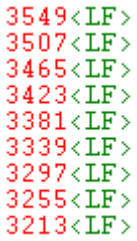


Figura 66: Posição enviada via RS232

Como é possível observar, os valores de posição têm intervalos constantes de 4,2º; tal indica que a velocidade é constante. Dado que a temporização do envio da posição angular é de 13 milissegundos, é possível então obter uma aproximação para a velocidade real.

$$velocidade_{rpm} = \frac{60000 \times 4{,}2}{13 \times 360} = 53{,}8\, rpm$$

Este cálculo de velocidade é baseado em períodos muito curtos, a possibilidade de obter valores dispares é maior, sendo que pequenas diferenças na temporização ou nos valores de posição induzem erros significativos. No entanto, o algoritmo o cálculo é diferente. Após completada uma revolução, o algoritmo averigua o intervalo de tempo decorrido desde a última revolução e calcula a velocidade média. No exemplo [Figura 67] que se segue pode-se calcular a velocidade média e compará-la com a enviada por linha série.

Figura 67: Envio de posição e velocidade média

Esta amostra apresenta a posição angular do LRF, sendo que caso sejam enviados dois valores na mesma linha o segundo é o referente à velocidade média. Neste caso a velocidade média é de 36 r.p.m.. É possível determinar a velocidade média contabilizando os valores enviados numa revolução, dado que estes são enviados com intervalos de tempo fixos é possível estimar o tempo decorrido.

$$Velocidade_{rpm} = \frac{60000}{126 \times 13} \approx 36{,}6\, rpm$$

A velocidade média calculada está próxima da velocidade real. Este exemplo serve também para mostrar que efectivamente se calcula a velocidade média, isto porque os intervalos entre posições variam, o que indica que também a velocidade variou, mas o algoritmo calcula a velocidade baseada num período de uma volta.

Ainda de referir que a rampa de aceleração criada correspondeu às expectativas, sendo que não se verificou nenhuma elevação súbita de velocidade ou falha na resposta do motor durante a mesma. Apesar de esta ser satisfatória, seria ainda possível melhorá-la ajustando o $TMR2_{presc}$ em função da frequência.

Conclusões e trabalhos futuros

Os resultados do controlo e supervisão da posição, velocidade e direcção são satisfatórios, na medida em que os valores obtidos estavam próximos do esperado e próximos do real.

O projecto é economicamente contido, sendo que o preço final restringe-se, aproximadamente aos 900 euros. De salientar que este preço não contempla o LRF e outros materiais já disponíveis no laboratório. Sistemas de laser existentes no mercado apresentam preços muito mais elevados. O sistema laser de alta definição Velodyne Erro: Origem da referência não encontrada custa cerca de 52800 euros, embora a qualidade não seja comparável, é possível assim ter uma ideia dos custos associados a este tipo de dispositivos.

A estrutura concebida é auto suficiente na medida em que não precisa de ser acoplada a nada para se manter estável durante a operação, esta é suficientemente pesada para que mesmo com o LRF em movimento a mesma se mantenha imóvel. Em contrapartida, embora móvel, a estrutura é algo pesada. A volumetria da estrutura parece torná-la mais indicada para reconstrução tridimensional, no entanto, com os devidos ajustes também pode ser utilizada em aplicações de condução autónoma.

O *slip ring* é uma peça fundamental para conseguir evitar os constrangimentos da cablagem, sendo que o projecto inicialmente previsto seria impossível sem esta peça. Isto possibilita o movimento contínuo, evitando assim a necessidade de ter fins de curso e ter constantes acelerações e desacelerações do LRF.

No futuro seria interessante estudar a possibilidade de ter unidades dedicadas só para a armazenamento da posição do LRF. Aplicando tal procedimento seria possível

averiguar se o algoritmo concebido estaria demasiado sobrecarregado com interrupts, e isso era um factor de perda de pulsos.

Poderia também se experimentar utilizar um microcontrolador com 4 entradas de *interrupt.* Assim seria possível ter uma rotina de *interrupt* associada ao canal B e averiguar se a contagem de pulsos proveniente do codificador se tornava mais robusta. Também seria interessante ter um *interrupt* associado ao sinal de controlo de motor, depois averiguar-se-ia qual seria a posição expectável do motor em função do número de pulsos enviados. Este processo também poderia ser feito através de uma unidade dedicada.

Seria também interessante analisar a viabilidade de criação de uma super unidade que recolhesse todos os sinais provenientes do codificador. Isto reforça a ideia anteriormente referida de criar uma unidade dedicada. Assim seria possível usar todos os 6 sinais do codificador: os três sinais normais e os três sinais invertidos.

Os sinais invertidos serviriam também para controlar a posição, sendo que, havendo dois sinais por canal, dai pudesse advir ainda mais fiabilidade no controlo de posição.

Referências

[1] J.L. Martínez et al,**"Progress in mini helicopter tracking with a 3D laser range-finder"**, IFAC Congress. Praha. (2005)

[2] Oliver Wulf and Bernard Wagner, **"Fast 3D scanning methods for laser measurements systems"**, International Conference on Control Systems and Computer Science, CSCS14, Bucharest, Romania (2003)

[3] Charles F. Bergh et al, **"A compact, low power two-axis scanning laser rangefinder",** The 7th Mechatronics Forum International Conference (2000)

[4] Christian Brenneke et al, **"Using 3D laser range data for SLAM in outdoor enviroments"**, IROS 2003, Las Vegas, EUA (2003)

[5] **"LMS 200 Laser measurement system – Indoor Version-"**, Catálogo da SICK

[6] Miguel Matos Dias, **"Scanner 3D para aplicações em modelação e navegação"**, Departamento de Engenharia de Mecânica da Universidade de Aveiro (2004)

[7] Rui P. R. Cardoso, Robertt A. F. Valente, Ricardo J. A. Sousa, **"Geometria de massas"**, Departamento de Engenharia Mecânica da Universidade de Aveiro (2005)

[8] **"PIC18F2455/2550/4455/4550 Data Sheet"**, Data sheet para PICs da Microchip, http://ww1.microchip.com/downloads/en/DeviceDoc/C18_User_Guide_51288j.pdf

[9] **"MPLAB® C18 C Compiler User's Guide"**, Manual para compilador C18 da Microchip, http://ww1.microchip.com/downloads/en/DeviceDoc/C18_User_Guide_51288j.pdf

[10] Simões Morais**, "Desenho de construções mecânicas",** Porto Editora (2004)

[11] **"Sanyo Denki – 103H7126-0440- Stepper Motor, 1.8 deg",** Manual de especificações do motor

[12] **Seattle Robotics Society**, http://www.seattlerobotics.org/guide/servos.html

[13] **Stepper Motor World,** http://www.steppermotorworld.com/introduction_and_Guide_to_Stepper_Motors.htm

[14] **Globalspec – How a slip ring works,** http://www.globalspec.com/reference/9665/How-a-Slip-Rings-Works

[15] Aiwu Zhang et al, **"Fast continuous 360 degree color 3D laser scanner",** Beijing

The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, ISPRS Congress Beijing (2008)

[16] Kai Pervölz *et al,* "**3D Laser Scanner for Tele-exploration Robotic Systems**", Proceedings of the International Workshop on Safety, Security and Rescue Robotics (Conferência: Gaithersburg, MD, E.U.A., Agosto 2006).

[17] "High Definition Lidar HDL-64E S2", Catálogo do LRF de alta definiçãoVelodyne, www.velodyne.com/lidar/products/brochure/HDL-64E%20S2%20datasheet_lowres.pdf

[18] "efunda – Sampling Theorem and Nyquist Rate", www.efunda.com/designstandards/sensors/methods/DSP_Nyquist.cfm

Anexos

A1. Main.c

```
////////////////////////////////////////////////////
//                                                //
//                   Main.c                       //
//    Departamento de Engenharia Mecânica         //
//           Universidade de Aveiro               //
//                                                //
//                 João Dias                      //
////////////////////////////////////////////////////


#include <p18f2455.h>
#include <string.h>
#include <stdio.h>
#include "C:\Users\João Dias\Documents\Dissertação\PIC Prog\30 Maio(09)\Func.h"
#include "C:\Users\João Dias\Documents\Dissertação\PIC Prog\30 Maio(09)\Var.h"

////////////////////////////////////////////////////
//                                                //
//                 INTERRUPTS                     //
//                                                //
////////////////////////////////////////////////////

#pragma code high_vector=0x08
    void interrupt_at_high_vector (void)    //interrupts de alta prioridade
    {
        _asm GOTO high_isr _endasm          //..."ir para" rotina high isr
    }
#pragma code

#pragma code low_vector=0x18
    void interrupt_at_low_vector(void)      //interrupts de baixa prioridade
    {
        _asm GOTO low_isr _endasm           //"ir para" rotina low_isr
    }
#pragma code

#pragma interrupt high_isr
    void high_isr (void)                    //rotina para interrupts de alta prioridade
    {

        if (INTCON3bits.INT1IF==1) {        //interrupt gerado pelo CANAL 0 (codificador)
            k*=217;                         //cálculo da velocidade
            vel=1E6;
            vel/=k;
            k=0;
            if (cw==0) {                    //se o sentido de rotação for anti-horário...
                deg1=0;                     //...levar os incrementos do codificador a 0
            }else if (cw==1) {              //se o sentido de rotação for horário...
                deg1=1024;                  //...levar os incrementos do codificador até 1024
            }
            if(pos_ref==1 && rs==1) {
                send(vel,'SameLine');
            }
            if(pos_ref==0 && RX==0) {
                pos_ref=1;                  //obtida posição de referência
            }
            INTCON3bits.INT1IF=0;           //o interrupt "apagado" por software
        }

        if (INTCON3bits.INT2IF==1) {                    //interrupt gerado pelo CANAL A (codificador)

            if (PORTBbits.RB3==1 && pos_ref==1) {       //se RB3 (CANAL B) já estiver activo
                cw=0;                                   //...então o LRF move-se no sentido anti-horário
            }else if (PORTBbits.RB3==0 && pos_ref==1) { //se RB3 (CANAL B) ainda estiver inactivo
                cw=1;                                   //...então o LRF move-se no sentido horário
            }
            if (cw==0 && pos_ref==1) {          //se o sentido de rotação for anti horário...
                deg1++;                         //...realizar 1 incremento posicional
            }else if (cw==1 && pos_ref==1) {    //se o sentido de rotação for horário...
                deg1--;                         //...realizar 1 decremento posicional
            }
            if (deg1>1024 && pos_ref==1) {      //se os incrementos forem superiores a 1024
                pos_ref=0;                      //...então perdeu-se a posição de referência
                SendPointer(Wrong_pos);
            }
            deg=deg1*35;                        //cálculo da posição angular
            deg/=10;
            if (deg1==Pos_stop && pos_ref==1){  //se a posição actual coincidir com a posição de paragem
                PWM_off();                      //...desactivar PWM
                send(deg,'NewLine');
```

```
            Pos_stop=4000;
        }
        pic_vel=5E6;                        //cálculo da velocidade definida no registo PR2
        pic_vel/=853;
        pic_vel/=PR2;

        INTCON3bits.INT2IF=0;       //o interrupt tem de ser "apagado" por software
    }
    if (INTCONbits.TMR0IF==1) {             //interrupt gerado pelo TIMER 0
        INTCONbits.TMR0IF=0;                //o interrupt tem de ser apagado por software
        TMR0L=0x06;                         //13 ms
        TMR0H=0xFC;
        k++;
        if(rs==1 && pos_ref==1) {           //se existir ordem de "verbose on"
            send(deg,'NewLine');            //...enviar a posição pela linha série
        }
    }
}

#pragma interruptlow low_isr
    void low_isr (void)                     //rotina para interrupts de baixa prioridade
    {
        if  (PIR1bits.RCIF==1) {            //interrupt gerado pela recepção de dados via RS232
            i=0;
            while (RCREG!='\r') {           //"enquanto" não chegar o caracter indicativo de fim da mensagem...
                str2[i]=RCREG;              //os caracteres são armazenados no array str2
                while(PIR1bits.RCIF==0);    //esperar pelo interrupt gerado pelo próximo caracter
                i++;
            }
            RX=1;                           //variável que indica que chegou uma mensagem pela linha série
            str2[i]='\r';                   //sinalização do fim do array
        }
    }
            ////////////////////////////////////////////////////
            //                                                //
            //               ROTINA PRINCIPAL                 //
            //                                                //
            ////////////////////////////////////////////////////

void main(void) {                           //rotina principal do programa

    PICinit();                              //definição inicial de parametros
    deg1=0;                                 //definição de algumas variáveis
    Pos_stop=4000;
    pos_ref=0;
    vel=0;
    rs=0;
    k=0;
    ss=0;
    vv=0;
    pp=0;
    RX=0;
    t=0;
    invalid_msg=0;

    if (start==1) {                         //se o PIC já foi iniciado...
        PICinit();
        SendPointer(resetstr);              //...então "fez" reset
    }   else if (start!=1) {                //se o PIC ainda não foi iniciado...
        start=1;
        SendPointer(initstr);               //...então foi iniciado
    }
    HomePos();
    SendPointer(home);

    while(1) {

            ////////////////////////////////////////////////////
            //                                                //
            //                  MENSAGEM                      //
            //                                                //
            ////////////////////////////////////////////////////

        if(RX==1) {
            RCSTAbits.CREN=0;
            pos_ref=0;
            j=0;
            instr_h=0;
            instr_v=0;
            instr_s=0;
```

```
instr_p=0;
instr_m=0;
instr_d=0;
invalid_msg=0;

    ////////////////////////////////////////////////
    //                                            //
    //                ANÁLISE MENSAGEM            //
    //                                            //
    ////////////////////////////////////////////////

while (str2[j]!='\r') {
    if ((str2[j]=='s' || str2[j]=='S') && instr_s==0) {     //velocidade
        n=0;
        j++;
        instr_s=1;
        while (str2[j]>=48 && str2[j]<=57) {
            j++;
            n++;
        }
        j--;
        if (n>3 || n<=0) {
            invalid_msg=1;
            break;
        }
    }else if ((str2[j]=='m' || str2[j]=='M') && instr_m==0) {   //movimento (desactivado)
        instr_m=1;
    }else if ((str2[j]=='b' || str2[j]=='B') && instr_d==0) {   //movimento sentido anti-horário
        instr_d=1;
    }else if ((str2[j]=='f' || str2[j]=='F') && instr_d==0) {   //movimento sentido horário
        instr_d=1;
    }else if ((str2[j]=='h' || str2[j]=='H') && instr_h==0) {   //homeposition
        instr_h=1;
    }else if ((str2[j]=='v' || str2[j]=='V') && instr_v==0) {   //verbose
        instr_v==1;
        j++;
        if (str2[j]>49 && str2[j]<48) {
            invalid_msg=1;
            break;
        }
    }else if ((str2[j]=='p' || str2[j]=='P') && instr_p==0){    //posição de paragem
        q=0;
        j++;
        instr_p=1;
        while (str2[j]>=48 && str2[j]<=57) {
            j++;
            q++;
        }
        j--;
        if (q>4 || q<=0) {
            invalid_msg=1;
            break;
        }
    }else {
        invalid_msg=1;
        break;
    }
    j++;
}

    ////////////////////////////////////////////////
    //                                            //
    //    PROCESSAMENTO E EXECUÇÃO DE MENSAGEM    //
    //                                            //
    ////////////////////////////////////////////////

if (invalid_msg==1) {
    SendPointer(Msg_corrupted);
} else if (invalid_msg==0) {
    SendPointer(Msg_received);
    j=0;
    m=0;
    p=0;
    while(j<i) {
        if (str2[j]>=48 && str2[j]<=57) {
            if (ss==1) {
                temp[m]=str2[j];
                m++;
                if (ss==1 && m==n) {
                    speed=Array2Int(temp,m);            //converter array em valor decimal
                    if (speed<=120 && speed>24) {
```

```
        instr_p=1;
        while (str2[j]>=48 && str2[j]<=57) {
            j++;
            q++;
        }
        j--;
        if (q>4 || q<=0) {
            invalid_msg=1;
            break;
        }
    }else {
        invalid_msg=1;
        break;
    }
    j++;
}

    ////////////////////////////////////////////////
    //                                            //
    //    PROCESSAMENTO E EXECUÇÃO DE MENSAGEM    //
    //                                            //
    ////////////////////////////////////////////////

if (invalid_msg==1) {
    SendPointer(Msg_corrupted);
} else if (invalid_msg==0) {
    SendPointer(Msg_received);
    j=0;
    m=0;
    p=0;
    while(j<i) {
        if (str2[j]>=48 && str2[j]<=57) {
            if (ss==1) {
                temp[m]=str2[j];
                m++;
                if (ss==1 && m==n) {
                    speed=Array2Int(temp,m);            //converter array em valor decimal
                    if (speed<=120 && speed>24) {
                        freq=speed;                     //conversão da velocidade em período
                        freq*=853;
                        Presc2=5E6;
                        Presc2/=freq;
                        SpeedRamp(Presc2,PR2);          //executar rampa de aceleração
                    } else {
                        SendPointer(Invalid_speed);
                    }
                    for (w=0;w<m+1;w++) {
                    temp[w]=' ';                        //limpar o array
                    }
                    ss=0;
                }
            }else if (pp==1) {
                temp[p]=str2[j];
                p++;
                if (p==q) {
                    Pos_int=Array2Int(temp,p);          //converter array em decimal
                    Pos_int*=10;                        //converter posição angular em pulsos
                    Pos_int/=35;
                    if (Pos_int<1024 && Pos_int>0) {
                        SpeedRamp(245,PR2);
                        Pos_stop=Pos_int;
                    }else {
                        SendPointer(Invalid_pos);
                    }
                    for (w=0;w<p+1;w++) {
                        temp[p]=' ';
                    }
                    pp=0;
                }
            }else if (vv==1) {
                if (str2[j]==49) {
                    rs=1;                   //activar envio de posição e velocidade
                }else if (str2[j]==48) {
                    rs=0;                   //desactivar envio de posição e velocidade
                }
                vv=0;
            }
        }
        if (str2[j]=='s' || str2[j]=='S') {
            ss=1;
        }else if (str2[j]=='v' || str2[j]=='V') {   //verbose
```

```
                }else if (str2[j]=='v' || str2[j]=='V') {   //verbose
                    vv=1;
                }else if (str2[j]=='p' || str2[j]=='P'){    //posição de paragem
                    pp=1;
                }else if (str2[j]=='m' || str2[j]=='M') {   //movimento (desactivado)
                    if (PORTAbits.RA0==1) {
                        SpeedRamp(245,PR2);
                        PWM_off();
                    }else if (PORTAbits.RA0==0) {
                        SendPointer(Stopped);
                    }
                }else if (str2[j]=='b' || str2[j]=='B') {   //movimento sentido anti-horário
                    if (PORTAbits.RA1==0 && PORTAbits.RA0==1) {
                        Speed=PR2;
                        SpeedRamp(245,PR2);
                        PWM_off();
                        PORTAbits.RA1=1;
                        wait(1000000);
                        SpeedRamp(Speed,PR2);
                    }else if (PORTAbits.RA1==0 && PORTAbits.RA0==0) {
                        PORTAbits.RA1=1;
                    }else if (PORTAbits.RA1==1) {
                        SendPointer(Invalid_rot);
                    }
                }else if (str2[j]=='f' || str2[j]=='F') {   //movimento sentido horário
                    if (PORTAbits.RA1==1 && PORTAbits.RA0==1) {
                        Speed=PR2;
                        SpeedRamp(245,PR2);
                        PWM_off();
                        PORTAbits.RA1=0;
                        wait(1000000);
                        SpeedRamp(Speed,PR2);
                    }else if (PORTAbits.RA1==1 && PORTAbits.RA0==0) {
                        PORTAbits.RA1=0;
                    }else if (PORTAbits.RA1==0) {
                        SendPointer(Invalid_rot);
                    }
                }else if (str2[j]=='h' || str2[j]=='H') {   //homeposition
                    SpeedRamp(245,PR2);
                    HomePos();
                    SendPointer(home);
                    k=0;
                }
                j++;
            }
            for (j=0;j<i+1;j++) {
                str2[j]=' ';    //limpar o array
            }
            SendPointer(Executed_msg);
        }
        RX=0;
        RS232();
    }
  }
}
```

A2. Func.c

```
                ////////////////////////////////////////////////////
                //                                                //
                //                    Func.c                      //
                //    Departamento de Engenharia Mecânica         //
                //           Universidade de Aveiro               //
                //                                                //
                //                  João Dias                     //
                ////////////////////////////////////////////////////

#include <p18f2455.h>
void PWM_on();
void PWM_off();

void wait(unsigned int value) { //temporizador

    long unsigned int i,k;
    for (i=0;i<400*value;i++) {
    k=k;
    }
}

void PWM_off() {
        PR2=0;
        CCP1CONbits.CCP1M3=0x0; //desactiva o gerador de pulsos
        CCP1CONbits.CCP1M2=0x0;
        CCP1CONbits.CCP1M0=0x0;
        CCP1CONbits.CCP1M1=0x0;
        wait(500000);
        PORTAbits.RA0=0x0;      //desactiva o enable do L297

}
void PWM_on() {
        PR2=245;
        CCP1CONbits.CCP1M3=0x1; //activa o gerador de pulsos
        CCP1CONbits.CCP1M2=0x1;
        PORTAbits.RA0=1;        //activa o enable do L297
}

void SendArray(char array[],int dim) {      //envio de arrays
        int s;
        for (s=0;s<dim;s++) {
            while (PIR1bits.TXIF==0);
            TXREG=array[s];
        }
}
void SendPointer(rom near char *nline) {    //envio de ponteiros

    while (*nline!=0) {
        while(PIR1bits.TXIF==0);
        TXREG=*nline++;
    }
}

                ////////////////////////////////////////////////////
                //                                                //
                //                  RS232                         //
                //                                                //
                ////////////////////////////////////////////////////

void RS232() {
    //rs232 comunicação assincrona, envio de caracteres
    TXSTAbits.TXEN=1;       //activar a escrita (Transmit enable)
    TXSTAbits.SYNC=0;       //activar modo assincrono
    BAUDCONbits.BRG16=1;    //activar 16 bits para o gerador de baud rate
    SPBRG=42;               //definição de baud rate de 116200 para uma Fosc de 20 MHz
    TXSTAbits.BRGH=1;       //selecção de modo baud rate elevado (high baud rate)
    TXSTAbits.TX9=0;        //negar nono bit, transmissão de 8 bits
    RCSTAbits.SPEN=1;       //activar porta série (serial port enable)
    RCSTAbits.CREN=1;       //activar recepção contínua
    RCSTAbits.RX9=0;        //negar nono bit, recepção de oito bits
    RCONbits.IPEN=1;        //activação dos níveis de prioridade para interrupts
    PIE1bits.RCIE=1;        //activação do interrupt para recepção de dados pela porta série
}

                ////////////////////////////////////////////////////
                //                                                //
                //         CONFIGURAÇÃO INICIAL DO PIC            //
                //                                                //
                ////////////////////////////////////////////////////
```

```
void PICinit() {

//definição de portas
    TRISB=0b11111111;       //definir todas as portas do registo B como entradas
    TRISA=0b1000000;        //definir todas as portas do registo A como saidas à excepção da porta 6

//definião das portas como digitais ou analógicas
    ADCON1bits.PCFG0=1;
    ADCON1bits.PCFG1=1;
    ADCON1bits.PCFG2=1;
    ADCON1bits.PCFG3=1;     //definir todas as portas A como sendo I/O digitais

//activação de interrupts
    INTCONbits.GIE=1;       //activa todos os interrupts (global interrupt enable)
    INTCONbits.PEIE=1;      //activa todos os interrupts periféricos (pheriferical interrupt enable)

    RS232();

//definição de diversos interrupts
    INTCON3bits.INT2IP=1;   //nível de prioridade elevada para o interrupt INT2
    INTCON2bits.INTEDG2=1;  //interrupt "on rising edge"
    INTCON3bits.INT1IP=1;
    INTCON2bits.INTEDG1=1;
    IPR1bits.RCIP=0;        //baixa prioridade para o interrupt RCIF
    INTCON3bits.INT2IE=1;   //activação do interrupt INT2
    INTCON3bits.INT1IE=1;   //activação do interrupt INT1
    INTCON2bits.TMR0IP=1;   //definir alta prioridade para interrupt de TIMER 0

//definição dos parâmetros para para PWM
    CCP1CONbits.DC1B1=0;    //os bits DC1B0:DC1B1 são os 2 bits menos significativos (LSBs)
    CCP1CONbits.DC1B0=0;    //do registo que permite definir o duty cycle
    CCPR1L=1;               //oito bits mais significativos (MSBs)
    TRISCbits.TRISC2=0;     //definir o resgisto RC2/CCP1/P1A como saída
    T2CONbits.TMR2ON=1;     //activar o timer 2
    T2CONbits.T2CKPS1=0x0;  //prescaler unitário
    T2CONbits.T2CKPS0=0x0;

//a porta RA1 é inicializada a zero para que o sentido de rotação seja definido como horário
    PORTAbits.RA1=0;
//a porta RA2 é activada para que esteja em funcionamento o mode "half step"
    PORTAbits.RA2=1;
//definição do timer0 que irá temporizar o envio das mensagens
    T0CONbits.TMR0ON=1;     //activação do timer0
    T0CONbits.T0CS=0;       //definir o timer0 como um clock interno
    T0CONbits.T08BIT=0;     //definir o contador para uma resolução de 16 bits
    T0CONbits.PSA=0;        //activar prescaler para o timer0
    T0CONbits.T0PS0=1;
    T0CONbits.T0PS1=0;
    T0CONbits.T0PS2=1;      //prescaler de timer 0 definido para 32
}

                ////////////////////////////////////////////////////
                //                                                //
                //                FUNÇÕES                         //
                //                                                //
                ////////////////////////////////////////////////////

void send(int count,char NL) {

unsigned int rest,lap1,lap2;
    int v=0,u,w;
    static char str[];

    if (NL=='NewLine') {
        while(PIR1bits.TXIF==0);
        TXREG=0xa;
    }else if (NL=='SameLine') {
        while(PIR1bits.TXIF==0);
        TXREG=0x20;
    }

    lap1=count;
    lap2=count;

    while (lap1>0) {        //contabiliza-se quantos algarismos tem o número
        v++;
        lap1/=10;
    }
    for (u=v-1;u>0;u--) {   //contabilizados os algarismos, inseri-los num array
        rest=lap2%10;
        rest+=0x30;
```

```
        str[u]=rest;
        lap2/=10;
    }
    str[0]=lap2+0x30;

    SendArray(str,v);       //enviar array por linha série
    }

int HomePos() {
        INTCON3bits.INT1IE=0;
        INTCON3bits.INT2IE=0;
        PWM_on();
        while(INTCON3bits.INT1IF==0);
        PWM_off();
        INTCON3bits.INT1IF=0;
        INTCON3bits.INT1IE=1;
        INTCON3bits.INT2IE=1;
}



int Array2Int(char cont[],int e) {      //conversão de array em decimal
    unsigned int x,r=0,a=1,r=0;

    if (e>4) {
        return 10000;
    }

    for(x=0;x<e-1;x++) {
        a*=10;
    }
    return r;
}


void SpeedRamp(int Pres,int OldPres){   //rampa de velocidades
    unsigned int rp,fraq;

    if (PORTAbits.RA0==0) {
        PORTAbits.RA0=1;
    }
    if (OldPres==0) {
        OldPres=245;
        PWM_on();
    }
    rp=OldPres;

    if (Pres>OldPres) {                 //Rampa de aceleração
        while (rp<Pres+1) {
            if (rp>150) {
                wait(5);
            }else if (rp>50 && rp<=150){
                wait(10);
            }else if (rp<=50){
                wait(20);
            }
            rp++;
            PR2=rp;
        }
    }else if (Pres<OldPres){            //Rampa de desaceleração
        while(rp>Pres-1) {
                if (rp>150) {
                wait(5);
            }else if (rp>50 && rp<=150){
                wait(10);
            }else if (rp<=50){
                wait(20);
            }
            rp--;
            PR2=rp;
        }
    }else if (Pres==OldPres) {
        PR2=Pres;
    }
}
```

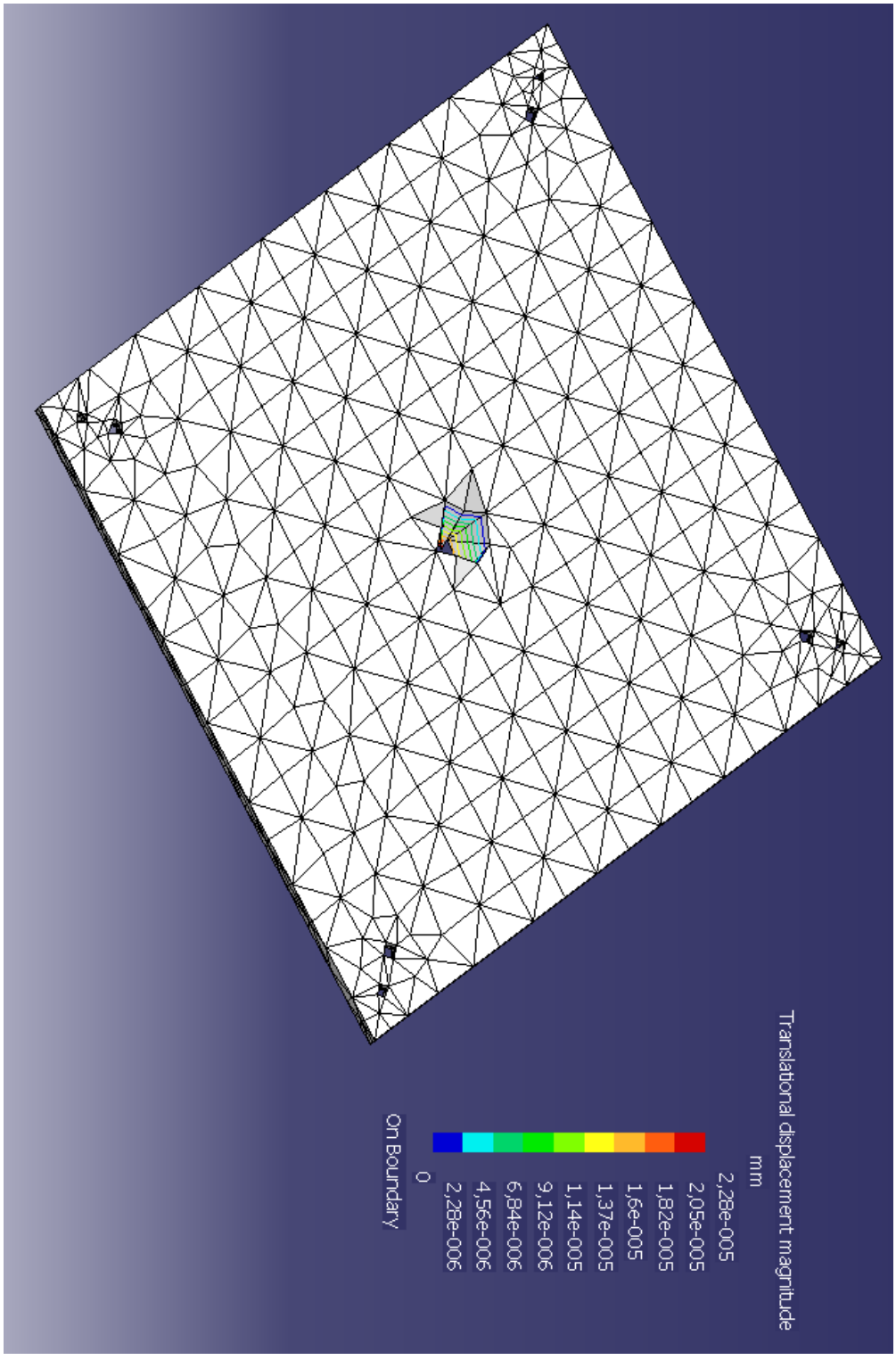
A3. Func.h

```
////////////////////////////////////////////////////
//                                                //
//                  Func.h                        //
//    Departamento de Engenharia Mecânica         //
//           Universidade de Aveiro               //
//                                                //
//                João Dias                       //
////////////////////////////////////////////////////

void PICinit();
void high_isr();
void low_isr();
void PWM_off();
void PWM_on();
void RS232();
int HomePos();
void wait(unsigned int value);
void SendArray(char array[],int dim);
void send(int count,char type);
void SpeedRamp(int Pres,int OldPres);
int Array2Int(char cont[], int e);
```

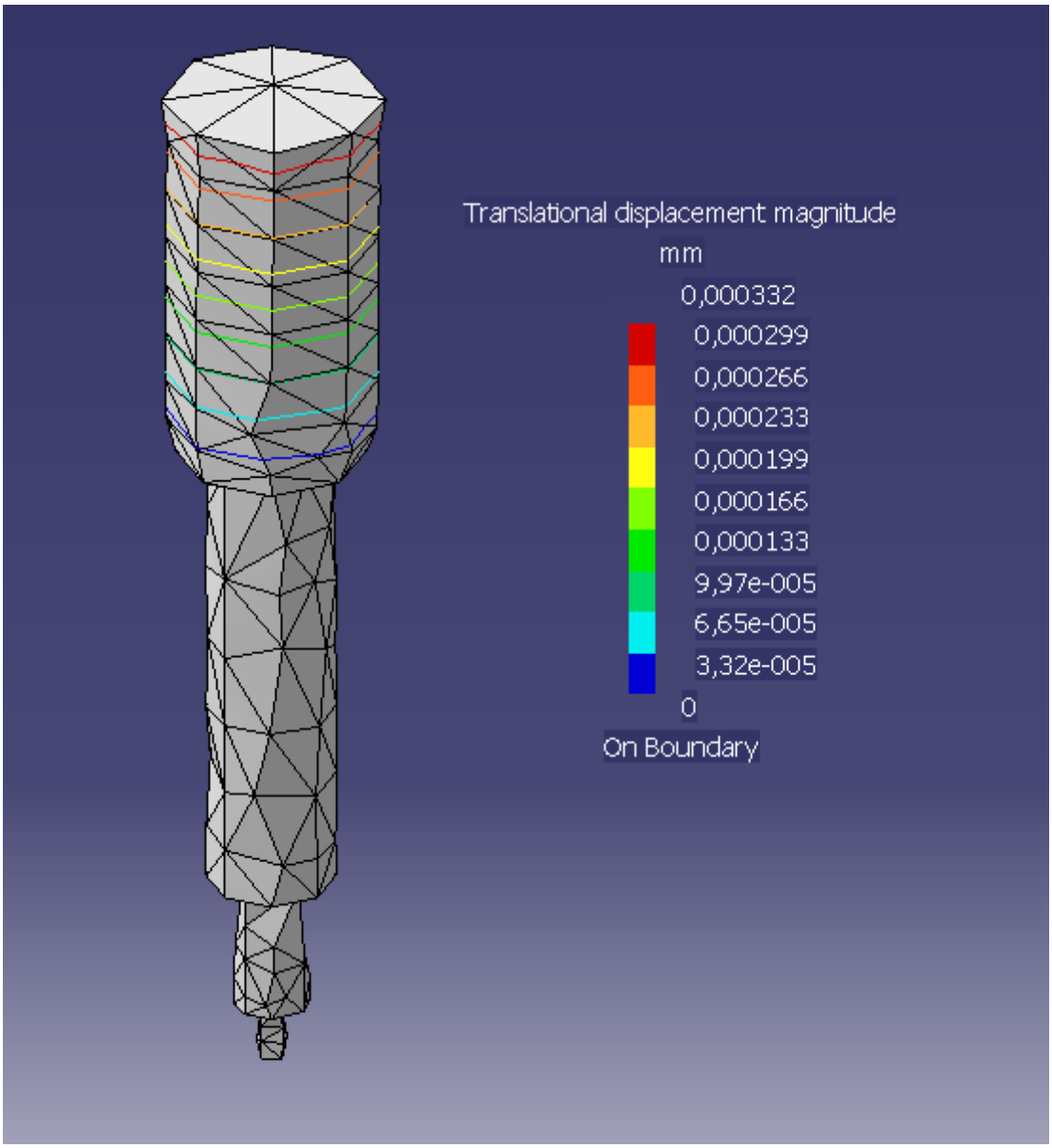
A4. Var.h

```
////////////////////////////////////////////////////
//                                                //
//                  Var.h                         //
//    Departamento de Engenharia Mecânica         //
//           Universidade de Aveiro               //
//                                                //
//                João Dias                       //
////////////////////////////////////////////////////

void PICinit();
int HomePos();
void PWM_off();
void PWM_on();
void RS232();
void wait(unsigned int value);
void SendPointer(rom near char *nline);
void send (int count,char type);
void SendArray(char array[],int dim);
int Array2Int(char cont[], int e);
void SpeedRamp(int Pres,int OldPres);
int invalid_msg,ss,vv,pp;
int start,cw,Pos_stop,rs;
unsigned int i,j,w,RX,t;
signed int instr_p,instr_s,instr_v;
signed int instr_h,instr_m,instr_d;
rom near char *initstr="\n \r inicializacao pic";
rom near char *resetstr="\n \r pic reset (MCLR) \n \r";
rom near char *home="\n \r home position  \n \r";
rom near char *Msg_received="\n \r Mensagem recebida \n \r";
rom near char *Msg_corrupted="\n \r Mensagem corrompida \n \r";
rom near char *Invalid_speed="\n \r A velocidade que inseriu é inválida \n \r";
rom near char *Stopped="\n \r O LRF já se encontra parado \n \r";
rom near char *Invalid_rot="\n \r O LRF já está a rodar nesse sentido \n \r";
rom near char *Executed_msg="\n \r Mensagem executada \n \r";
rom near char *Not_responding="\n \r O motor não está a responder correctamente \n \r";
rom near char *Wrong_pos="\n \r Posições incorrectas, inibida a transmissão \n \r";
rom near char *Invalid_pos="\n \r A posição que inseriu é inválida \n \r";
char temp[],str2[];
unsigned int T2Pres,NewSpeed,deg,deg1;
unsigned int pos_ref,speed,m,n,p,q;
unsigned long int Presc2,freq,k,vel,pic_vel;
unsigned short long int Speed,Pos_int;
```

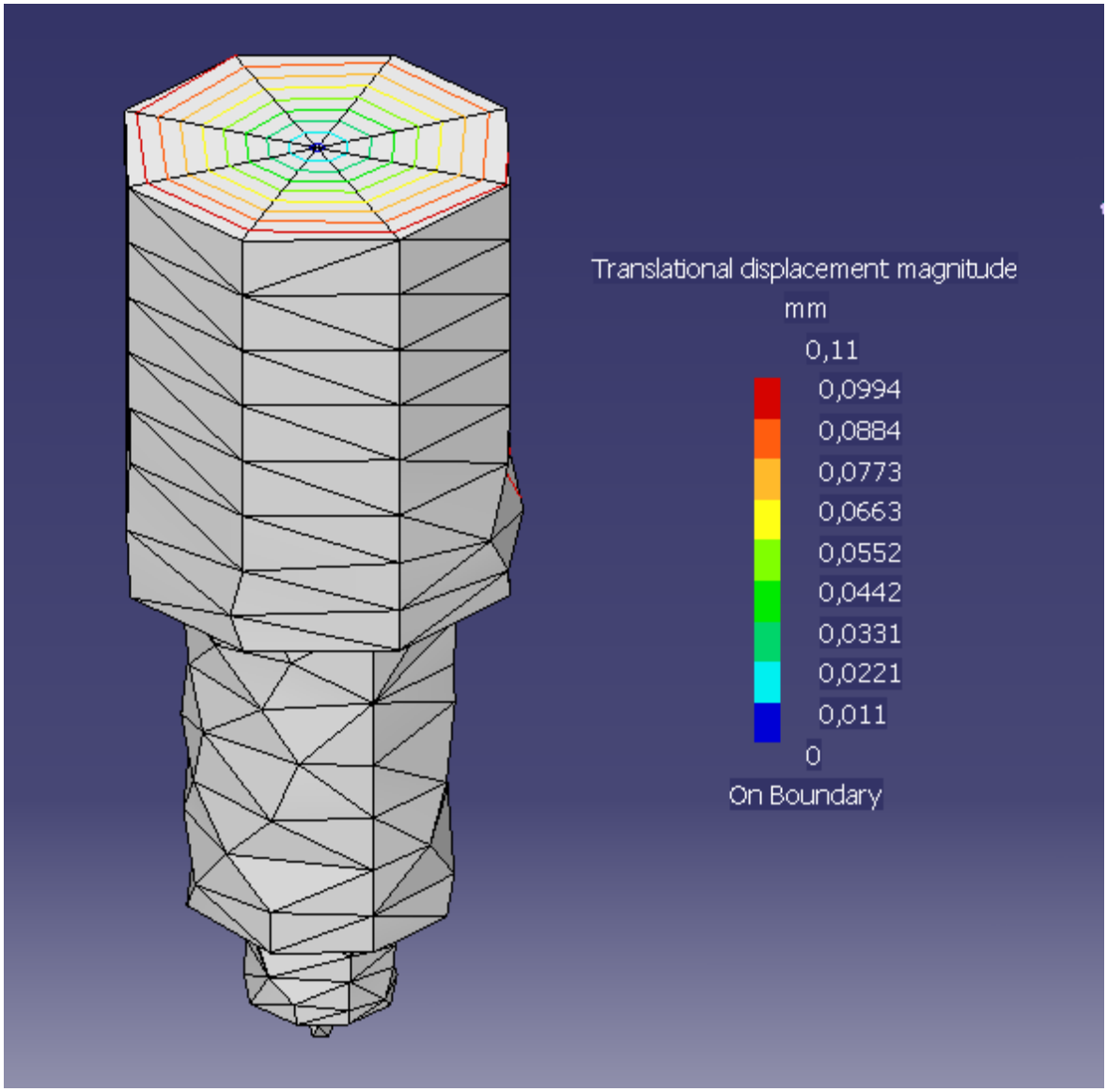
A5. Chapa de topo em alumínio – simulação de compressão (100 N)



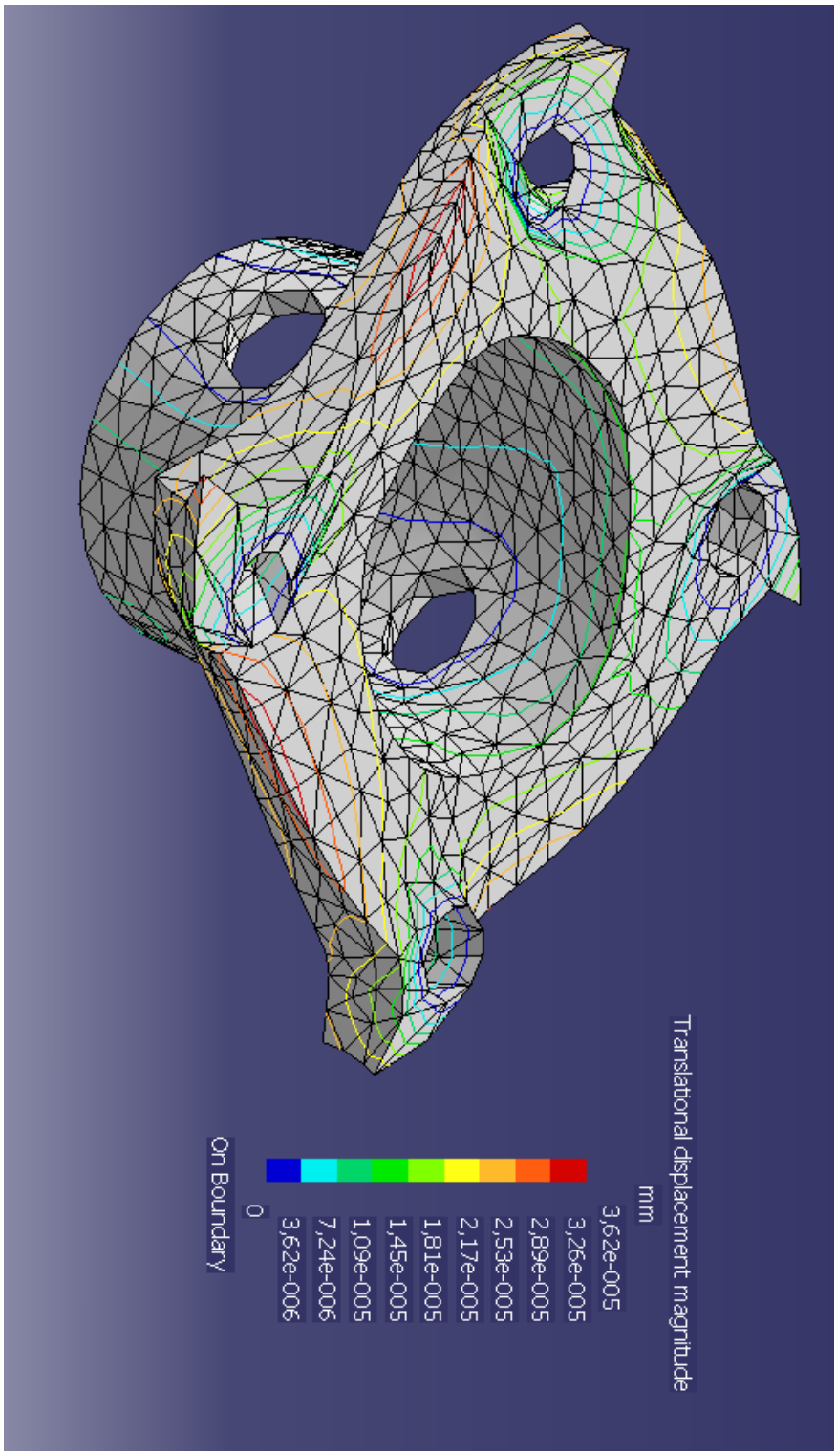
A6. Eixo central em alumínio – simulação de compressão (100 N)



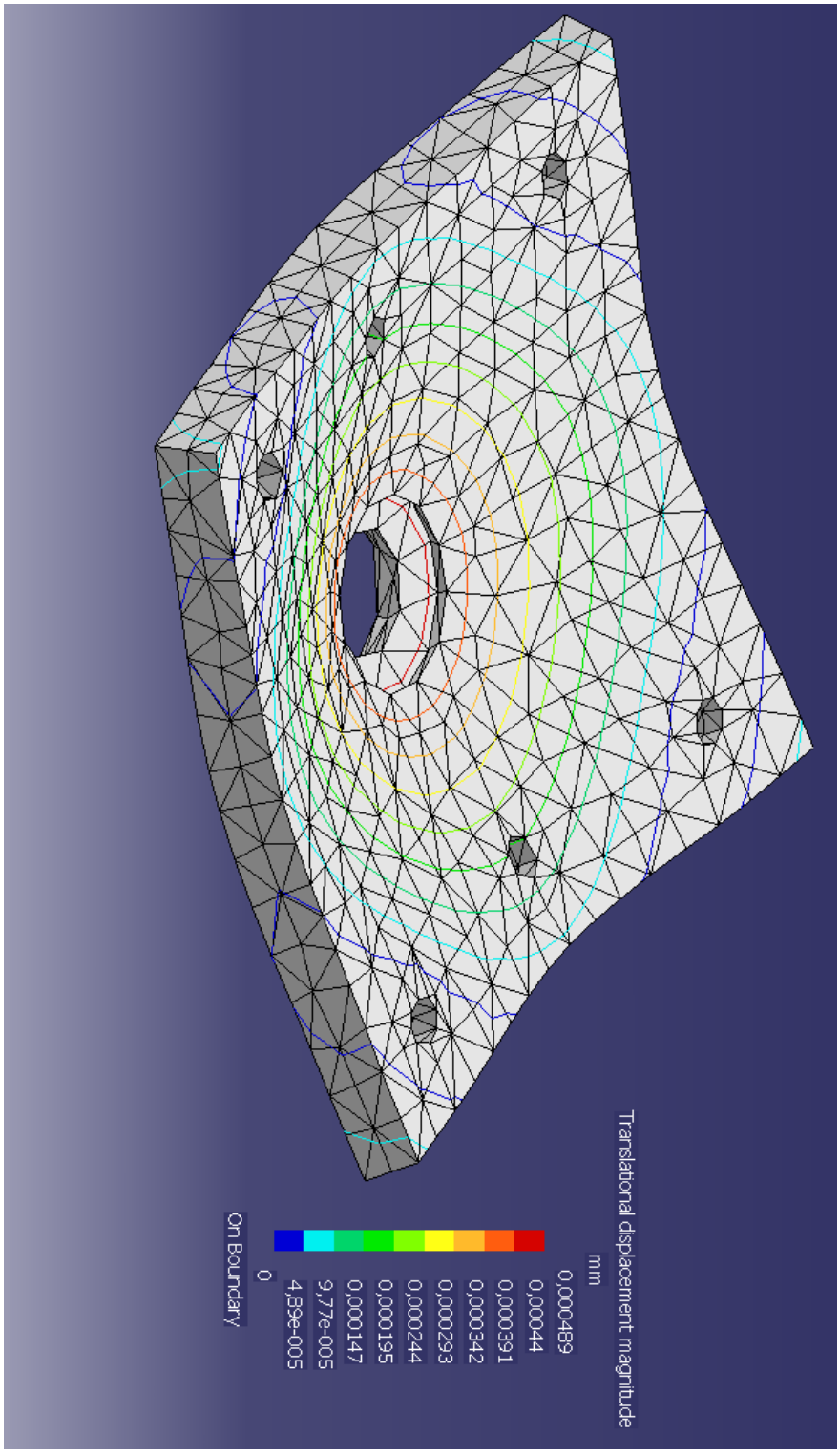
A7. Eixo central em alumínio – simulação de torção (1,27 N.m)



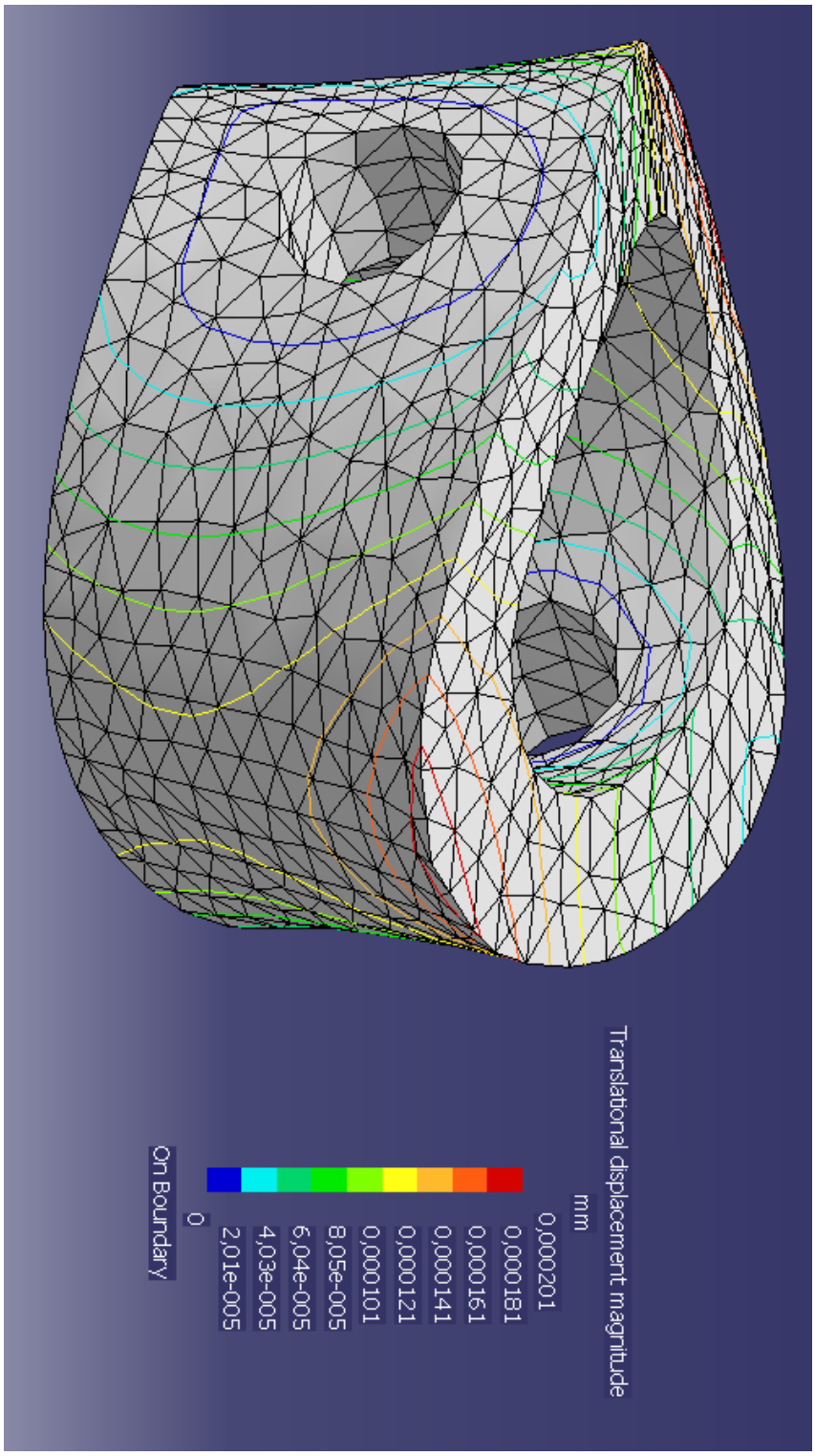
A8. Fixador em alumínio – simulação de torção (1,27 N.m)



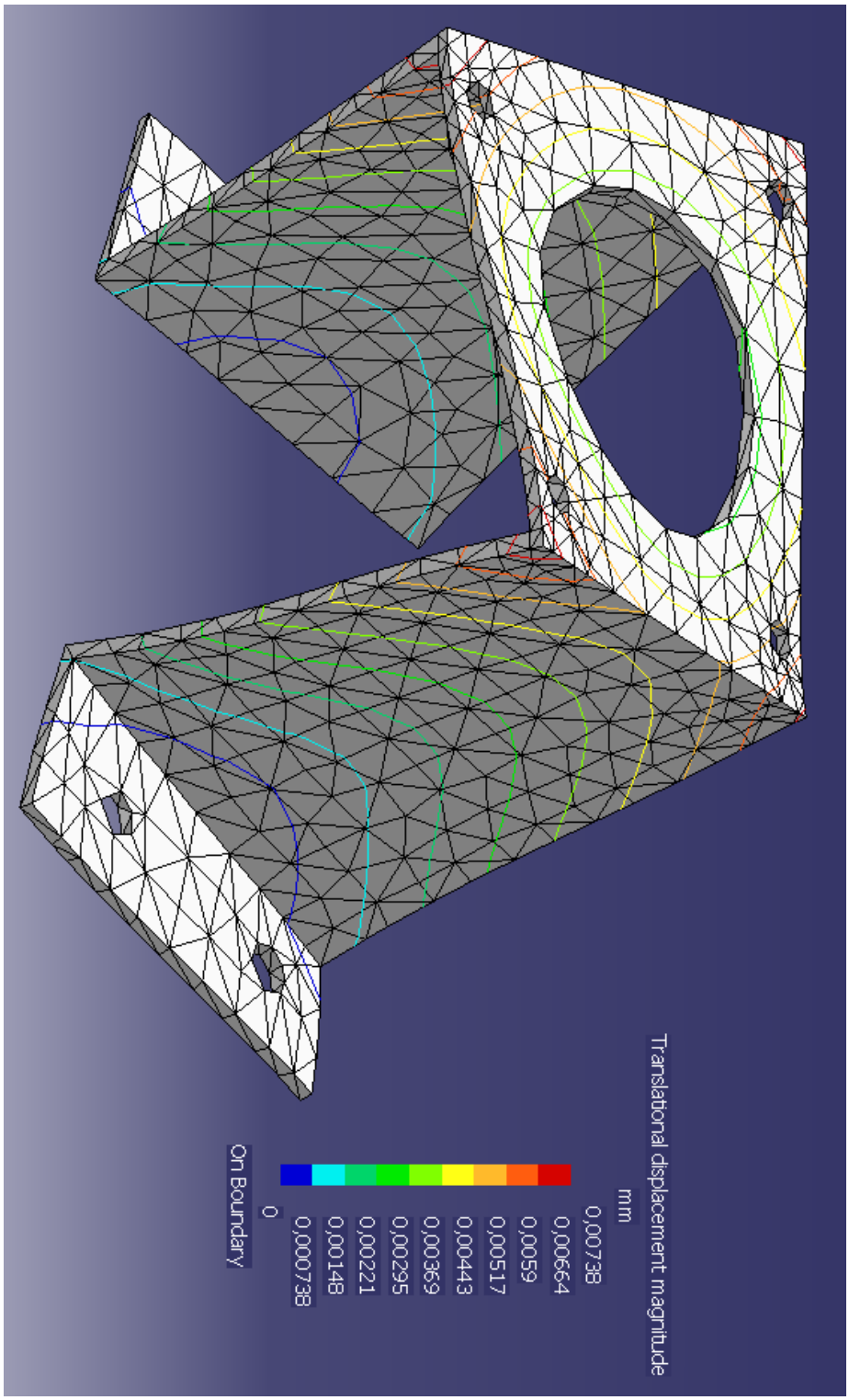
A9. Suporte para eixo em alumínio – simulação de compressão (100 N)



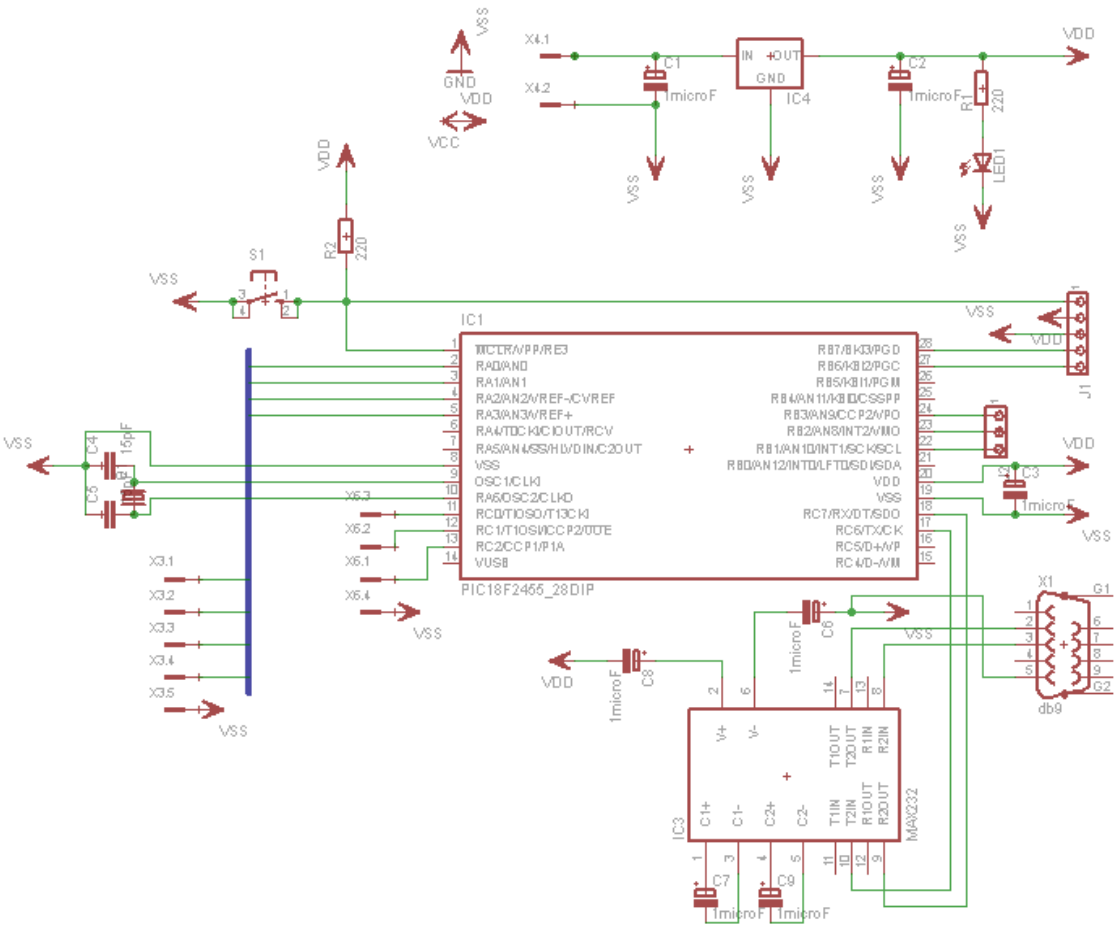
A10. Suporte inversor em alumínio – simulação de compressão (100N)



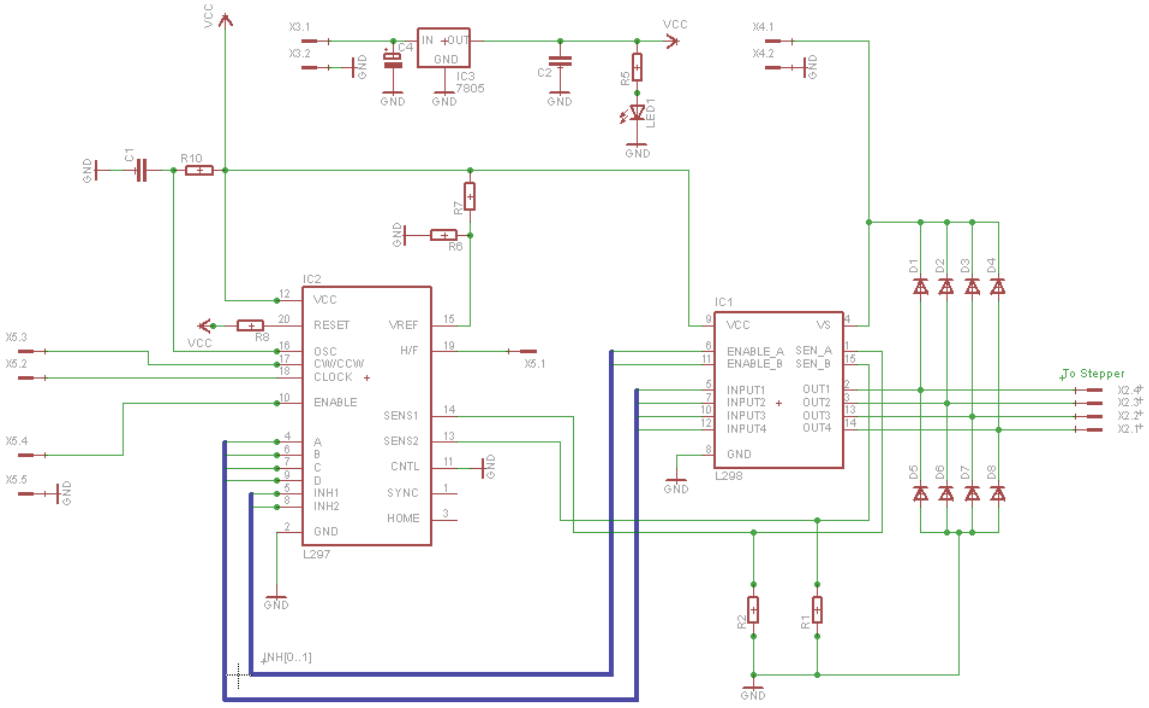
A11. Suporte para motor em alumínio – simulação de torção (1,27 N.m)



A12. Circuito microcontrolador – esquema eléctrico



A13. Circuito potência (L297+L298) – esquema eléctrico



A14. Circuito de alimentação e divisor de frequência – esquema eléctrico